68 MICRO JOURNAL

Australia A $13.00 New Zealand NZ $19.70
Singapore S $29.90 Hong Kong H $48.00
Malaysia M $28.50 Sweden 60.-SEK

$5.90 USA

OS-9 Systems Integration

6800 6809 68008 68000 68010 68020 68030

The Magazine for Motorola CPU Devices *For Over a Decade!*

A User Contributor Journal And Lots More!

VOLUME XII ISSUE I ● Devoted to the 68XXX User ● Jan./Feb. 1990

"FOR THOSE WHO NEED TO KNOW"

000422 A/E MJ
MR. MICKEY FERGUSON
P. O. BOX 87
KINGSTON SPRINGS TN 37082

PHOTO CREDIT: NASA

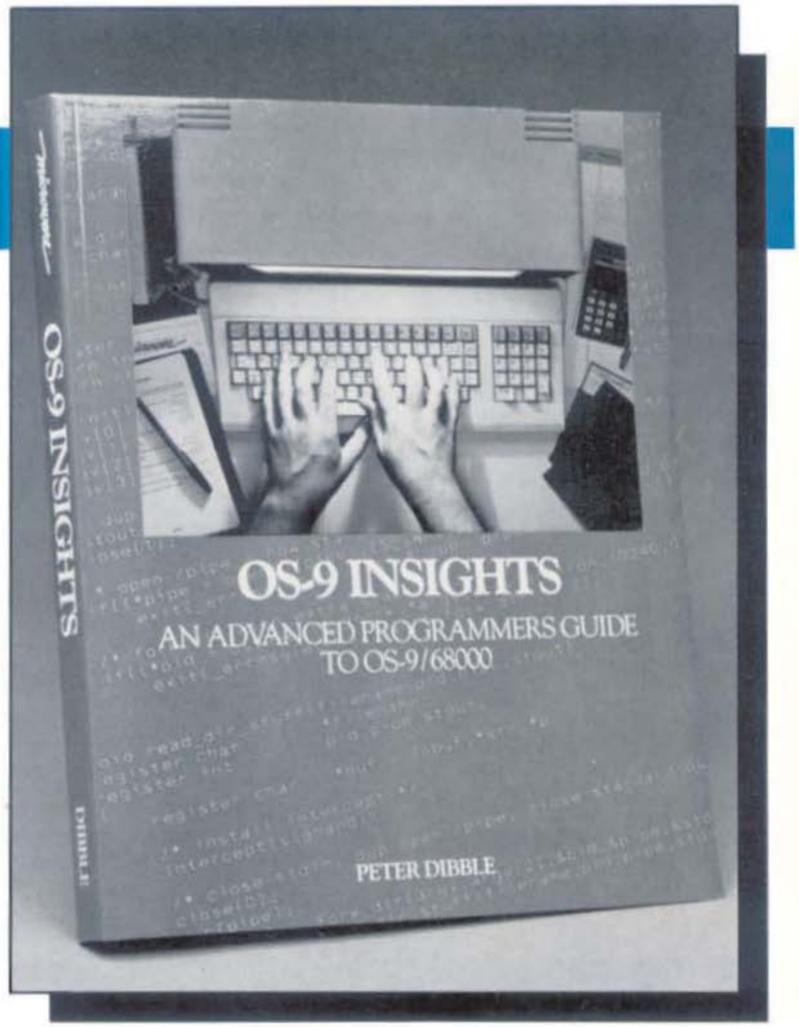
AVAILABLE NOW FROM MICROWARE!

OS-9 INSIGHTS

An Advanced Programmers Guide To OS-9/68000

- An in-depth examination of the OS-9 design philosophy
- Detailed discussion of Kernel operation and real-time features
- Sample file manager and driver source listings
- Information on customizing your OS-9 system
- An invaluable tutorial for the professional programmer

"This book was written for programmers who would like to use the advanced features of OS-9. It explains and illustrates features of OS-9 ranging from memory management through file managers."
—Peter Dibble



This is a "must" book for all serious OS-9 programmers!

Use this handy order form

Name ______________________
Address ______________________
City ______________________
State ______________ Zip ________

Please send me ______ copy(s) of OS-9 Insights at $40.00 each. Add $2.00 shipping and handling for each copy ordered.

TOTAL (includes shipping) $ ____________

Fill in all the information on this order form. Mail the completed order form along with your check, money order (no cash, please) or credit card information to:

microware®

MICROWARE SYSTEMS CORPORATION

1900 N.W. 114th Street • Des Moines, Iowa 50322
Attention: Order Department
Or Call: 515-224-1929

Credit card customers must complete the following:
(Check One)

☐ MasterCard ☐ VISA

Credit Card # ____ ____ ____ ____

Expiration Date ____/____

Signature ______________________
(Required — Credit Card Customers Only)

DON'T DELAY — ORDER TODAY!

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	5.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.2*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
        {
                register long i;
                for (i=0; i < 999999; ++i);
        }
```

Estimated MIPS - MUSTANG-020 ..... 4.5 MIPS,

Burst to 8 - 10 MIPS: Motorola Specs

Mustang Specifications

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
buffered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

CPI
Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead!
ONLY Data-Comp delivers the Super MUSTANG-020

OS-9

OS-9 Professional Ver	$850.00
*Includes C Compiler	
Basic09	450.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Omegasoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00

Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adapter Cards below RS232 Adapter	165.00
Each card supports 4 additional ser. ports (total of 36 serial ports supported)	
60 line Parallel I/O card	398.00
Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	
Prototype Board	75.00
areas for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	
SBC-AN	475.00
Interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call	
LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

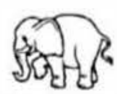
Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

The P R O !

Only the "PRO" Version of OS-9 Supported!



This is *HEAVY DUTY* Country!

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
	Total:	$5299.80

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

NEW LOWER PRICES

25 Mbyte HD ~~$4299.80~~ $3749.80

85 Mbyte HD ~~$5748.80~~ $4548.80

Note: Only Professional OS-9 Now Available (68020 Version) Includes ($500) C Compiler - 68020 & 68881 Supported - For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division

"A Decade of Quality Service"

Systems World-Wide

CPI Computer Publishing, Inc. 5900 Cassandra Smith Road Hixson, Tn 37343

Telephone (615) 842-4601 - FAX (615) 842-7990

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

The Originator of "DeskTop Publishing™"

6800 6809 68000 68010 68020

Toll Free Subscription Line 1-800 669 6809

Subscription Rates

USA

1 Year $ 49.00, 2 years $ 79.00, 3 years $ 99.00

Canada & Mexico, USA funds

1 Year $ 58.00, 2 Years $ 95.00, 3 Years $ 125.00

Other Countries

Surface Rates, USA funds

1 Year $ 61.00, 2 Years $ 95.00, 3 Years $ 125.00

Air Mail Rates, USA funds

1 Year $ 97.00, 2 Years $ 150.00, 3 Years $ 225.00

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World

Wide"

68 MICRO JOURNAL

Computer Publishing Center
5900 Cassandra Smith Road
Hixson, TN 37343
Phone (615) 842-4600 FAX (615) 842-4607

68 Micro Journal is the *original* "DeskTop Publishing" product and has been continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. There were no commercially available DTP packages! Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal (ISSN 0194-5025) is published 12 times a year by Computer Publ'ishing Inc. Second Class Postage paid at Hixson, TN. and additional entries. POSTMASTER: send address changes to 68 Micro Journal, POB 849, Hixson, TN 37343.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. Diagrams are acceptable, but please no hand written articles or blue ink.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

Contents

Features

Columns & Contributions

68 MICRO JOURNAL

68 Micro Journal™ Editorial

Welcome to the 1990's

From the Desk of Larry E. Williams

The staff and management of 68 Micro Journal hopes that all of you enjoyed a happy new year and that next year will be a most prosperous year for each of you. It's been a very fast pace for us here at 68 Micro Journal in the last quarter of 1989, and I'd like to use a few lines here to bring you up on all the exciting changes 68 Micro Journal has and will be taking.

68 Micro Journal is now an OS9 Systems Integration Magazine. We have spent many hours of consultation with various groups including developers, design engineers, researchers and others to determine the best possible target group for 68 Micro Journal to focus on. The overwhelming response was to highlight OS9 the operating system because of its extensive penetration into several promising markets that are keys to the growth of Motorola and other chip manufacturers for the future. The level of response to 68 Micro Journal in Europe over the past year has been testament to that. There has been many requests from overseas that 68 Micro Journal has not been positioned to properly respond to. This change of direction was made with input from many sources, both domestic and overseas.

With the Aug./Sept. 1989 issue we took the first step in bringing to the magazine, articles that were of interest to a greater number of our new subscribers and older ones as well. We warmly welcome input from our subscribers as well as others. We encourage you to write or call and tell us what you feel would be the best subject matter for future articles. We hope these changes will be beneficial to our readers as well as our advertisers.

With all the changes we were making over the past few months it was not possible to publish each month. It is our goal to be again publishing monthly beginning with the May 1990 issue. Subscription labels with the March/April issue will indicate the adjustment in your expiration date. The subscription rate that you paid was for 12 issues and

therefore all expiration dates will reflect some changes since we had 2 combined issues in 1989 and will have two in 1990 and we did not publish an Oct., Nov. or Dec 1989 issue.

The changes you see in the yearly subscription rates have taken place for several reasons. 68 Micro Journal will pay for OS9 articles. Articles submitted for pay must be clearly marked as such and submitted on an OS9 formatted disk. Write or call for an authors guideline sheet if you are interested. These changes are necessary to allow 68 Micro Journal to remain as the professional OS9 users magazine.

OS9 is clearly a leader in real time operating systems around the world today, and for that reason 68 Micro Journal will keep step to bring its readers a higher level of useful articles and information for the OS9 market place. The past decade has been exciting and full of changes, and we here at 68 Micro Journal are proud to have been there as those changes took place.

Since it takes as long as 90 days to reach a large number of our overseas subscribers, we will accept from overseas subscribers through the month of May 1990 subscriptions at the old rate of $36.50 per year surface and $72.50 per year airmail overseas. This extension of the old rate will only be available to current overseas subscribers. 68 Micro Journal will accept from current USA subscribers the old subscription rate of $24.50 per year through the end of February 1990. Take advantage of this while you can.

Without ever having received any advertising support from Motorola it has been difficult to stay committed to just being a Motorola magazine. However, 68 Micro Journal still feels that Motorola makes the best microprocessor in the world and we will continue so support Motorola in the future with coverage. However, as OS9 is introduced and developed on other platforms 68 Micro Journal will be right there covering OS9 as the best real-time operating system in the world.

Last but not least we want to thank all of you who have loyally hung in there all these years. And, as in the past, let us know what you want and what you feel we should be doing. After all it is still our readers who determines what we will be.

OS-9 for PERSONAL COMPUTERS

By : *Erich L. Gibbs, Ph.D.,*
President of Gibbs Laboratories, Inc.,
Wilmette, IL

AN ASSESSMENT

Sharing information is one of the most important functions a computer can provide, yet most of the millions of computers in the world are stand-alone, personal computers, which support only one user at a time. Single-user computers of any kind represent an incredible waste of computing potential. With few exceptions, they idle away well over 90% of their potential, waiting for users to make decisions, or simply waiting between keystrokes on the keyboard. A computer can accept keyboard entries about 50 times faster than anyone can type. No one disputes the fact that personal computers have contributed positively and extensively to revolutionizing the way man thinks and works; however, the cost has been unnecessarily high. As a result, productivity has not been as great as it might have been, and the benefits of computer technology have not been as widely distributed as they might have been. Gibbs Laboratories, Inc., has had the good fortune to be part of a number of high-technology breakthroughs during its 50 years of existence; we realize how difficult it can be to protect a technology from becoming distorted by marketing pressures. From our vantage point, we do not see the long term wisdom to flooding the market with single-user, strictly graphics-oriented computers to the exclusion of multi-user, multi-tasking capability. Designers and manufacturers had alternatives; they could have offered the market low-cost, multi-user, multitasking personal computers, which provided a little less *whizz* and a lot more *bang*.

SUMMARY

Worldwide, tens of millions of personal computers are on the verge of becoming competitively obsolete and slipping from the market, because they lack multi-user, multi-tasking capability. The market does not have the capital or sophistication to upgrade more than a small percentage of these installations using network technology or by replacement with new, high-performance personal computers, stuffed with memory. The powerful OS-9 operating system is memory efficient, has a high comfort index with software developers, and enjoys an excellent reputation for support. OS-9 would revolutionize the market, if it could be easily installed into personal computers. The Ultrasciencedivison of Gibbs Laboratories, Inc., has developed practical, low-cost products for adding OS-9 to IBM-style PC/XT/AT and Macintosh computers without the loss of existing applications, warranties, or support.

"This is an exciting time for OS-9, and we are thrilled to be part of it."

Personal computers could have been designed to support a graphics monitor as well as serial devices. Bit-mapped graphics plays an important, much needed, role; however, serial devices make more efficient use of computer hardware, because they can use compressed ASCII computer code to transfer information. Filling the screen of a serial, business terminal (often referred to as "dumb" video display terminal) requires transmitting about 20,000 bits of data down the data hookup wires. Creating exactly the same screen on a standard monochrome graphics monitor takes about 20 times as many bits. A color monitor requires about 200 times as many bits. Of all the tasks performed by computers, only a relatively small percentage are really bit-mapped graphics dependent. Desktop publishing is graphics dependent, but word processing is not; neither are spreadsheets, account-

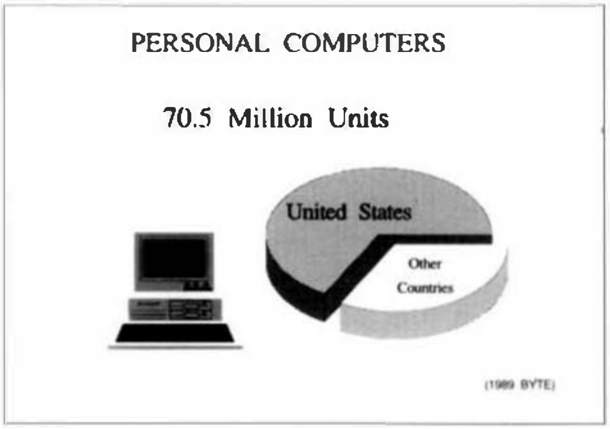
ing, electronic mail, automatic data acquisition, machine control, ... The market cannot afford to do everything in bit-mapped graphics mode; it wants to run serial terminals, bar code readers, electronic cash registers, business machines, instruments, production equipment, and a host of other devices in a multi-user, multi-tasking environment - inexpensively!

A few years ago, our Ultrascience division took a long, hard look at the personal computer market and began intensive research and development on products designed to adapt personal computers to be the practical, multi-user, multi-tasking computers they could have been in the first place. We sensed that most of the existing personal computer market would never be able to bear the cost and complexity of LANs and other solutions which require discarding current hardware for high-end computers stuffed with memory. Productivity/cost ratios of new personal computer systems were increasing, but so were up-front costs. As a result, the sales of each new round of systems were forcing a relatively greater number of existing systems into non-competitive obsolescence. More and more individuals and businesses were discovering that they were working to support their computers, rather than the other way around. Our goal was to contribute products which would help to reverse this trend and increase the learning and earning power of people, worldwide.

THE OS-9 SOLUTION

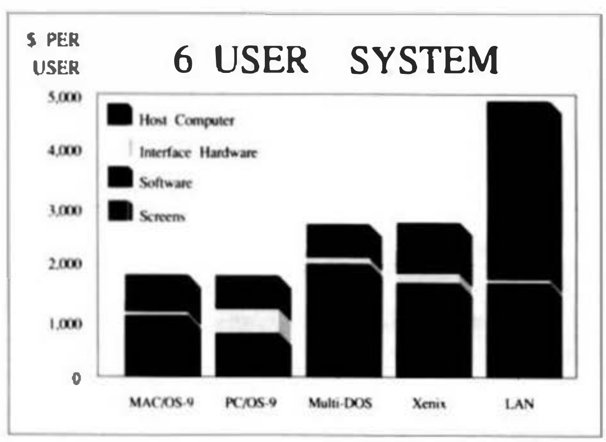
Given the scope of the undertaking, we had no illusions. If we were to be successful, we would have to develop products which would take advantage of existing market forces and the enormously complex worldwide supply and support structure. We could not redirect the market; our products would have to serve a catalytic function by removing obstacles, preventing the market from accomplishing what it was already inclined to do.

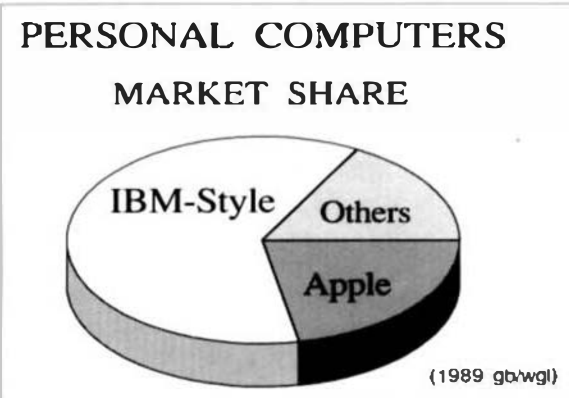
Software is the most expensive component of most computer installations. It would be impossible to persuade a generation of personal computer users to consider an enhanced approach, if they first would have to give up what they already had. Our multi-user, multi-tasking products would have to run in conjunction with existing personal computer software. Also, it would be impossible to implement any solution which would not make maximum use of the existing base of programming experience and programming tools. There would be neither time nor money to extensively retrain and retool enough programmers. What we needed was a very powerful, multi-user, multi-tasking operating system which would work efficiently in small memory spaces. It would have to be full-featured, support the major languages, and provide today's programmers with a high comfort level and room to grow. The operating system would also have to be hardware independent, so that software written for personal computers could run on larger and faster computers without reprogramming, thereby ensuring a smooth, practical upgrade path.



The OS-9 operating system from Microware was exactly what we required, and more. Though not yet a household name, OS-9 enjoys a well deserved, worldwide reputation for performance, and it is utilized by such industry giants as Sony and Philips. The computer industry knows OS-9 as ***the*** original operating system for the most powerful and widely used family of general purpose computer processor chips available, the Motorola 68K family.

OS-9 combines significant new operating system concepts and real-time capabilities with the overall architecture of the popular UNIX operating system; yet, the core software for Professional OS-9 requires only a tiny fraction of the memory required for UNIX, less than 150Kb. Without sacrificing programming convenience and features, OS-9 achieves exceptionally fast context switching times and interrupt response. The versatile memory module design is re-entrant and position independent; the system requires only one copy of a program, no matter how many people may be using that program. OS-9 supports arithmetic and I/O coprocessors, graphics, networking, and *standard* versions of all the major software languages, BASIC, C, FORTH, FORTRAN, MODULA-2, MUMPS, PASCAL, etc., and it is hardware independent. Powerful system-state, user-state, and language debuggers are available. There is a cornucopia of office automation, data bases, and applications available for OS-9 computers, and the list is growing rapidly. OS-9 versions of UNIX tools and utilities, such as *curses* and the *Bourne Shell*, are available to make conversion of UNIX applications convenient. *Curses* neutralizes the differences between different brands of terminals, so application programmers can write standardized code for windowing and display en-





hancements (i.e. blinking, underline, reverse video, etc.). The *Bourne Shell* is a powerful command programming language. It can be used to build individual programs, even those written in different languages, into complex, interactive, high-level applications.

The market dictated which personal computers we would initially support. IBM-style PC/XT/AT (compatible brands and models included) and Macintosh computers are well defined standards and, together, they represent the lion's share of all those computers presently in operation. The diversity of third party products available for these computers and the strength of their service/support systems contributed significantly to their success; it was imperative that our products install without modification of standard, compatible hardware or software. In fact, we would have to go a step beyond. It would be necessary to interface the standard DOS and MAC software environment gracefully from OS-9; otherwise, it would be impractical to support the wealth of existing third party hardware.

OS-9 FOR IBM

Since OS-9 is designed for use with the Motorola 68K family of processors, it cannot run on the processor chips built into IBM-style personal computers. Therefore, the Ultrascience solution for installing OS-9 into IBM-style personal computers required designing an inexpensive, but powerful,68K coprocessor board, which would be compatible with the PC, XT, or AT hardware and software environment. The PC68K1, as it is called, is a complete stand-alone 10 or 12.5 MHz computer with 1Mb of zero wait-state DRAM, two efficient, vectored serial ports, a parallel port, battery backed up time-date chip, and an optional battery backed up 8Kb static RAM (see specifications for details). It communicates with the DOS bus by means of a high-performance dual ported 1Kb DRAM. A companion serial expansion board, the MEMIOX, provides up to 10 more vectored serial ports and additional zero wait-state DRAM (see specifications for details). The PC68K1 and optional MEMIOX simply plug into free bus slots in any IBM compatible PC, XT, or AT computer (even compatible laptop computers have been used). The DOS operating system is left undisturbed and all PC/XT/AT devices are accessed from OS-9 by means of DOS calls. This means that OS-9 can support any standard, PC/XT/AT compatible third party device. Moreover, warranty and support for the PC/XT/AT will be unaffected by the installation of OS-9.

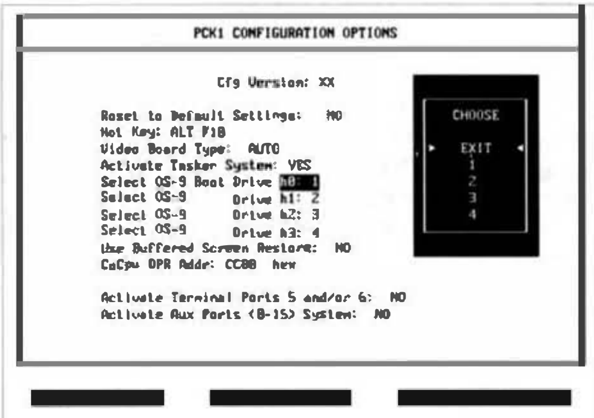
Multi-User IBM system running OS-9

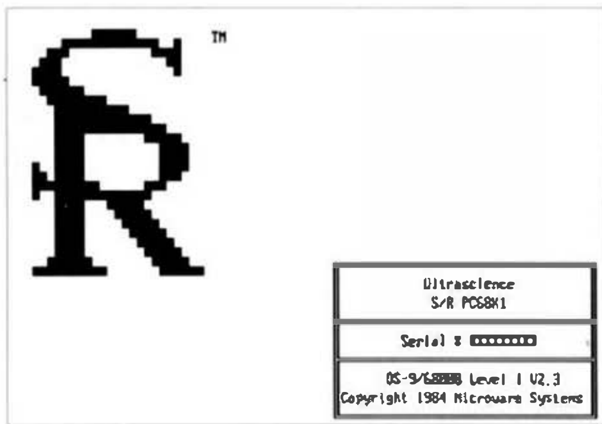
Powerful on-board diagnostics, which can identify defects down to specific chips, are performed each time the PC68K1 is reset. The diagnostic report is printed from the first PC68K1 serial port.

An extensive collection of diagnostic software tools are also provided for cases of suspected intermittent problems. All chips are socketed, so that a PC68K1 or MEMIOX can be easily repaired, on-site or locally, without the need for even a soldering iron.

Installation of a PC68K1 and OS-9 into an existing system usually takes less than an hour. The manual provides step-by-step instructions, which are complemented with plenty of screen grabs and pictures. First, a partition is added to one or more of the hard disks. Menu-driven PC68K1 system setup software makes it convenient to tailor an OS-9 system for many hardware configurations; OS-9 partitions can be installed on any of up to 4 hard disks and any of these partitions can be designated as the boot partition. Once the OS-9 partitions have been designated, the software can be loaded.

A few DOS programs are copied from a diskette into a DOS directory on the





hard disk, using an *INSTALL* program. Next, OS-9 is booted from a diskette, a hard disk partition is high-level formatted for OS-9, and the OS-9 software is loaded onto the hard disk by means of an *install* program. This completes the installation and the PC68K1 equipped PC/XT/AT is ready to boot OS-9 and run real-time, multi-user, multi-tasking applications from the monitor and any or all of its ports.

The monitor can be hot-keyed back and forth between DOS and OS-9, while OS-9 is running on from 2 to 12 PC68K1/MEMIOX serial ports and the PC68K1 parallel port. An excellent, very full featured OS-9 terminal emulating driver supports monochrome and color monitors. When a color monitor is used, programs designed for monochrome serial terminals will automatically display in color without program modifications.

DOS and OS-9 share PC/XT/AT compatible serial and parallel port(s) as well as floppy, hard disk, and tape drive(s), with the exception that unintelligent "floppy" interface tape drives have proven too inefficient to be practical. PC68K1 software dynamically optimizes time sharing of DOS resources within limits that can be set manually or under program control. Descriptors and drivers support the Microware UV-580, 5807, and 5407 5¹/₄" diskette formats and the UV-380 and 3807 3¹/₂" diskette formats. Data and programs can be copied back and forth between DOS and OS-9. DOS programs can be spawned from OS-9 and live data can be passed between OS-9 and DOS programs.

OS-9 FOR MACINTOSH

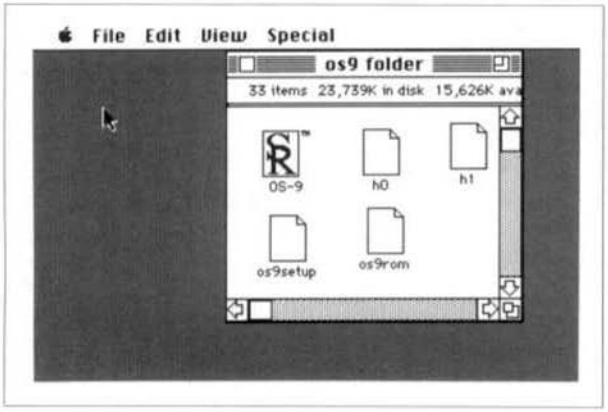
Macintosh computers are designed around Motorola68K processorchips, so we were able to install OS-9 without the need for any hardware. OS-9 and the Macintosh operating system operate together on the same 68K processor chip. Since the Macintosh operating system is left undisturbed and Macintosh devices are accessed from OS-9 by means of Macintosh Toolbox calls, OS-9 can support any standard, Macintosh compatible, third party device. Moreover, warranty and support for the Macintosh will be unaffected by the installation of OS-9.

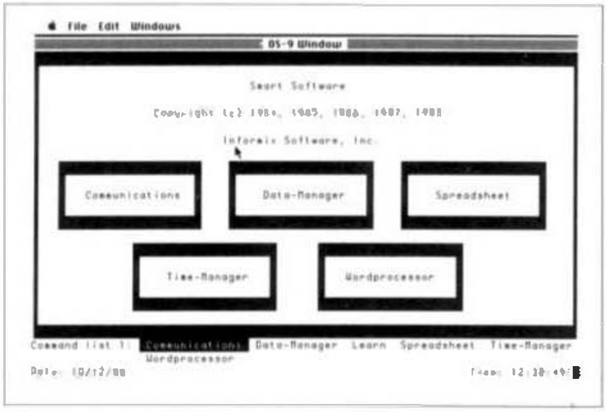
Installation of OS-9 usually takes less than half an hour. The MAC and OS-9 software are provided on MAC/800K diskettes. To install the software, the user creates a folder (directory) called "OS9 folder" and "drags" (copies) all of the diskette files into the folder. Once the software is loaded, selecting "OS9" will boot OS-9 and the Macintosh is ready to run multi-user, multi-tasking applications from the monitor and any or all of its ports. Battery backed up time and date are available to OS-9 from the Macintosh. OS-9 system setup software provides the means to specify the amount of memory available for OS-9 and the size of up to 8 OS-9 hard disks, including the bootable hard disk. In the Macintosh environment, OS-9 "hard disks" are actually Macintosh files. This in no way alters the operation of OS-9; however, it does permit OS-9 "hard disks" to be backed up with any Macintosh backup device.

OS-9 supports both color and b/w Quick Draw commands for the Macintosh monitor, giving OS-9 for the Macintosh significant graphics capability. As in the case of the IBM implementation of OS-9, there is an excellent, very full featured OS-9 terminal emulating driver for standard Macintosh screens and the larger add-on color or black and white monitors. When a color monitor is used, programs designed for monochrome serial terminals will automatically display in color without program modifications.

The practical limit to the number of users on a Macintosh, running OS-9, will depend on the model. A Mac-Plus is a 68000 machine, which runs at 8MHz. It will support a few users. On the other

Multi-User MAC system running OS-9





end of the spectrum, a Mac-IIcx uses a 68030 processor running at 16 MHz and it will support 15 or more users. A number of serial expansion products are available for Macintosh computers. We have tested the Hurdler[1], an intelligent 4 port serial board for the Nu-Bus, and performance was excellent. Adding extra OS-9 ports is as simple as connecting expansion hardware and installing driver software.

The capability of running Macintosh applications concurrently with OS-9 applications is presently under development and is scheduled for release during the first quarter of 1990. Support for a number of standard Microware diskette formats, using Macintosh Superdrive compatible disk drives, will be offered at the same time.

THE UPGRADE PATH

OS-9 for personal computers frees the user from the expensive hardware upgrade path normally associated with personal computers. Many personal computer users do not realize that VMEbus computer systems, using high-performance 68030 processors, are in the same price range as fast personal computers, yet VMEbus systems can support a hundred or more serial ports. When a more powerful computer becomes justifiable, everything that was invested into the OS-9 adapted personal computer system can be salvaged. Changeover could be completed in minutes, because OS-9 is hardware independent and the PC/XT/A Tor Macintosh could be connected to the larger computer as a workstation/terminal. Ultrascience offers DOS and Macintosh operating system equivalents of the OS-9 terminal emulating driver for the monitor. These programs make it possible for DOS or Macintosh computers to hot-key between their normal operation and operation as a terminal for a computer running OS-9.

[1] Creative Solutions Inc., 4701 Randolph Rd., Suite 12, Rockville, MD 20852

CONCLUSION

It took a dedicated team of hardware and software engineers to design and produce practical products, which permit OS-9 to be installed into IBM-style PC/XT/AT and Macintosh personal computers. We had our share of false starts and blind alleys; however, we are confident we have created catalytic products which will make multi-user, mutli-tasking operation of personal computers a practical, and profitable reality for both the user and applications developer. Our products for installing the OS-9 operating system into personal computers are easily used and inexpensive. Users will be able to boost productivity without a major capital outlay. OS-9 will permit applications programmers to open new markets and reestablish fading markets with more powerful products that are more easily developed and supported. This is an exciting time for OS-9, and we are thrilled to be part of it.

Desktop VMEbus System

68030 capable of supporting up to 100 ports

OS-9 is a trademark if Microware Systems Corp.
IBM is a trademark of IBM Corp.
Mac is a trademark of Apple Computer, Inc.
S/R is a trademark of Ultrascience
UNIX is a trademark of AT&T Bell Laboratories

C

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Laua Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter discusses typedef declarations and provides a comment about real compiler-compilers.

It also poses a problem and provides a file-compare program.

TYPEDEF DEFINITIONS

The typedef declaration is one of the lesser-used and more misunderstood elements of the C language.

It allows the programmer to create new names for existing data types, although it does not allow the programmer to create new data types.

In its simplest form, a typedef declaration is similar to a #define declaration.

Thus,

```
typedef int size;
```

may be used almost interchangably with

```
#define size int
```

in contexts such as

```
size i, j, k;
```

which would be equivalent to

```
int i, j, k;
```

except that typedef declarations are local to the block in which they are defined and #define declarations are global to the source file in which they appear, applying to all source code physically following them in the same compilation.

More complex typedef declarations are less similar to #define declarations, however, which may lead to problems of readability.

Thus,

```
typedef int * size;
```

may not be used interchangably with

```
#define size int *
```

in contexts such as

```
size i, j, k;
```

since the typedef is equivalent to

```
int *i, *j, *k;
```

and not equivalent to

```
int *i, j, k;
```

as implied by the #define declaration.

Following are some examples of the use of typedef declarations, as taken from actual programs.

From a Tic-Tac-Toe program:

```
typedef struct
[
int value; /* The move returned by the */
int path;  /* alphabeta consists of a value */

] MOVE; /* and an actual move (path) */

/* Alphabeta: takes a board */
MOVE alphabeta(board, whosemove, alpha,
beta)
int board[]; /* whose move, alpha&beta
cutoffs, */
int whosemove; /* and returns a move to
make and */
int alpha;    /* the value that the move has */
int beta;
[

MOVE result, successor;
int best_score, i, best_path, mademove;

}

main() /* The actual game */
{
int i, board[9];
char ch;
MOVE ourmove;
{

}
```

From the Dhrystone program:

```
typedef enum
{

Ident1, Ident2, Ident3, Ident4, Ident5

}    Enumeration;

struct Record
{

struct Record *PtrComp;
Enumeration Discr;
Enumeration EnumComp;
OneToFifty IntComp;
char StringComp[30];

};

typedef struct Record * RecordPtr;
```

```
typedef struct Record RecordType;

RecordPtr PtrGlb;
RecordPtr PtrGlbNext;

PtrGlbNext =
(RecordPtr)malloc(sizeof(RecordType));
PtrGlb =
(RecordPtr)malloc(sizeof(RecordType));
PtrGlb->PtrComp = PtrGlbNext;
PtrGlb->Discr = Ident1;
PtrGlb->EnumComp = Ident3;
PtrGlb->IntComp = 40;
strcpy(PtrGlb->StringComp, "DHRYSTONE
PROGRAM");
```

Is the following sequence of recursive typedef declarations legal with respect to either the original K & R definition of the C language or with respect to the new ANSI standard for the C language?

```
typedef char byte;
typedef unsigned byte u_byte;
```

The answer is apparently negative in both cases, despite the fact that some current compilers attempt to accept such syntax.

One argument against it is that the following sequence is not clearly defined by either standard:

```
typedef int (*pfi)();
typedef unsigned pfi upfi;
```

and because of this ambiguity, it is or should be explicitly non-syntactical.

As a further argument, the following sequence of typedef declarations is ambiguous, invalid, or both:

```
typedef char CHAR;
typedef unsigned char UCHAR;
typedef signed char SCHAR;

typedef unsigned CHAR Char
typedef unsigned UCHAR UnsignChar;
typedef unsigned SCHAR SignChar;
```

If all of the last three typedefs above are not illegal, which are legal and which are illegal?

It might be much easier on a compiler to use the same representation for CHAR and SCHAR on a signed char machine, and the same representation for CHAR and UCHAR on an unsigned char implementation.

The original set of typedefs could be restated as follows:

```
typedef signed char byte;
typedef unsigned char ubyte;
typedef char utiny;
```

As to whether an ANSI-compliant C compiler should issue syntax errors on such ambiguous declarations, the ANSI C standards committee's response would probably be "that is a quality of implementation issue".

Thus, a C compiler which correctly compiles conforming source but does something bizarre, such as emitting bad code, for non-conforming source could still be called an ANSI-compliant C compiler. Despite the best efforts of the standards committee, there will always be ambiguous aspects of the C language which will be processed differently by various C compilers.

COMPILER-COMPILERS

Bob Knighten, of Encore Computer Corporation, 257 Cedar Hill Street, Marlborough, Maine 02172, (508-460-0500 ext. 2626), provided the following comments about production compiler-compilers.

My first was the SofTech ALS Ada compiler for the Vax. I worked on this towards the end of the implementation phase. This was a fairly direct implementation of the technique described in Glanville's thesis. Graham was actually a consultant during the early design stages for this compiler. Glanville didn't really address global optimization/register allocation issues and the ALS compiler dealt with this by doing on-the-fly register allocation based on lifetime information produced by the global optimizer which worked on a largely machine independent expanded DIANA tree.

The parser generator was written by a SofTech engineer. It was written in Ada, the parsers were in Ada and the parser tables were Ada aggregates. A minor but important change in the language accepted by the parser (compared to Glanville's design) was to factor addressing modes. This was crucial to bringing the size of the parse tables for the VAX down to merely LARGE.

This compiler was retargeted to various of the Intel microchips (8086, 80286, ?) and to a navy computer (AN/UYK-?). The local code generation was acceptable, but for the VAX and AN/UYK the code quality was poor. The reason for this was that the register management scheme really did not work at all well. The code quality for the Intel machines was fairly good. This was partly because register management is just not very important and partly because more tuning was done on the compilers.

The SofTech code generator was redesigned (by a group headed by me and a colleague) for the NTT/DIPS machine. This is a variation on the IBM 370 architecture. Instead of using the straight LR approach of Graham and Glanville, we went to an affix grammar approach. This is similar too, but not the same as Ganapathi's approach. For this we contracted with MetaWare to modify their TWS to produce an affix grammar parser. The actual language used is described in the MetaWare TWS manual (such as it is). It shows a notable Ada influence. Again the parser tables are Ada aggregates, and the parser skeleton is written in Ada.

We also changed a number of aspects of the code generation/register allocation process. The machine description used was not for the actual machine, but rather for a similar virtual machine having all three operand instructions and an infinite supply of all types of registers. [In contrast to the IBM machine, the DIPS has separate arithmetic and address registers.] Following code generation, we then do flow analysis, replacement of three operand instructions by two operand instructions and register binding using a register coloring approach.

I left SofTech before the implementation was complete, but I've been assured by friends who are still there that the quality of code generated is quite good. We actually had a contractual commitment to produce code within some fixed percentage of the execution speed of the best hand assembled code for a set of benchmarks and we met that goal.

All of these compilers are quite slow. This was primarily because the front end and global optimizer are slow, but this just allowed the code generator to slip by. These are slow code generators, but this seems to be a reflection that speed was not important rather than an inability to be fast.

The most recent CGG based compiler I was involved with is the Retargetable Back End at Prime Computer. Aspects of that project are discussed in two recent papers:

Prescott K. Turner, "Up-down parsing with prefix grammars", SIGPLAN Notices, Vol. 21 No 12 (Dec. 1986) pp. 167-174

David Spector and Prescott K. Turner, "Limitations of Graham-Glanville style code generation", SIGPLAN Notices, Vol. 22 No. 2 (Feb. 1987) pp. 100-108

The major change here was that Scott Turner came up with a different parsing method. I'm told that Weisgerber and Wilhelm at the Universitat des Saarlandes have devised a similar tree matching code generator.

Two compilers were built at Prime using this technique for code generation. They were used internally, but never released due to a change in corporate plans. The speed was not good and the code quality was poor. Again the local code quality was quite good, but nothing was done take advantage of global information. In particular the problem of integrating register management well had not been solved when the project was cancelled.

The Intermetrics AIE Ada compiler as being based on PQCC technology. Most of the recent Intermetrics compilers have been inspired by the PQCC project and the Ada compiler came much closer to using the full PQCC approach, but it's probably too strong to say it was based on the PQCC technology.

This also illustrates why the term "Code generator generator" is ill-advised. The original hope of the PQCC project was to largely automate the production of compilers using as input language and machine specifications. At Intermetrics a large suite of tools have been built to help automate the building of compilers, but the process has not gotten terribly much faster or cheaper. Similarly at SofTech, the reason for going with the Graham-Glanville technique was to produce retargetable compilers. But the process of retargetting these compilers has not proved particularly easier than building hand crafted code generators from well-defined specifications. The reason for this is that local code selection - the part that can be done by parsing, tree matching, what have you - is not the hardest part of code generation, so there remains a substantial hand crafted part.

There are other production compilers based on CGGs:

Rodney Farrow, "Experience with an attribute grammar-based compiler" SIGPLAN Notices, 1982

This is a report on Intel's Pascal-86 compiler.

Hans-Stephan Jansohn, Rudolf Landwehr, Jochen Hayck and Michael Thamer, "Generating MC68000 code for Ada", SIGPC Notes, Vol. 6 No. 2 (1983) pp. 81-87 (1983 ACM Conference on Personal and Small Computers)

This is a discussion of the University of Karlsruhe Ada compiler which is based on the GAG (Generator based on Attribute Grammars) and CGSS (Code Generator Synthesis System). At least the VAX version of this compiler is available from a small German company, Systeam A.G.

John Yates and Robert Schwartz, "Dynamic programming and industrial-strength instruction selection: Code generation by tiring, but not exhaustive, search", SIGPLAN Notices, Vol 23 No. 10 (October 1988) pp. 131-140

This describes the new code generators being used in the Apollo compiler for their new RISC machines. It is based on bottom-up tree matching and was implemented using LEX, YACC and C code.

Finally I mention that COMPASS has an automated code generator production system (PEARL?) based on a lexical analysis technique. This is discussed in a COMPASS technical report by Kathleen Knobe and Joan Lukas.

Based on my experience, both direct and indirect, I've come to feel that the value of machine generated code generators (or, more likely, machine generated code selectors) is in providing a framework for generating AND MAINTAINING many different compilers. And this not because making one new compiler is much faster this way than in a less automated fashion, but because this enforces a discipline and uniformity which is very hard to achieve without automation.

C PROBLEM

A reader submitted the following problem.

```
/*

Given a file named tab.dat like the following:

I 11 16 88 22 20 10
O 11 16 88 22 23 21
I 11 16 88 22 23 30
O 11 16 88 22 23 39

When I do the following:

cc -g filename.c
sdb
r

I get a "doscan.c no such file or directory"
error from sdb.

This snippet of code works when I read
character
values from the file

(ie: fscanf(fp, "%s %s %s %s %s %s %s",

c1, c2, c3, c4, c5 ,c6, c7)

but it does not work when I try to read integer
values from the file with the fscanf in the
code.

Any ideas why valid integers from a file can't
be
read as integers but can be read as char
strings?

*/

#include <stdio.h>

main()
{

FILE *fp;
char a[80];
int i;
int q;
int r;
int s;
int t;
int u;
int v;

fp = fopen("tab.dat", "r");
for (i = 0; (i < 3); ++i);
{

fscanf(fp, "%s %d %d %d %d %d %d",

a, q, r, s, t, u, v);

printf("%s %d %d %d %d %d %d\n",

a, q, r, s, t, u, v);

}

}
```

There are at least two errors in this program.

The first should be obvious from the comments, but the second is more subtle.

Since the scanf family of functions must return values, its arguments must be addresses, not values. This is why char string arguments work correctly in the program above, but integer arguments do not work. The fscanf function should have been coded as the following:

```
fscanf(fp, "%s %d %d %d %d %d %d",
```

a, &q, &r, &s, &t, &u, &v);

However, when this problem is corrected, the program runs and does not encounter an address error, but only reads and outputs the first line of the input file.

The problem is the extra semicolon at the end of the for statement, which causes the null statement between the close parenthesis and the semicolon to be executed three times, but the fscanf-printf group to be executed only once. Thus, this statement should be restated as follows:

for (i = 0; (i < 3); ++i)

EXAMPLE C PROGRAM

Following is this month's example C program: it compares two files.

```
#include    <stdio.h>
#include    <ctype.h>

#ifdef IBMPC
#define BINARY "rb"
#endif
#ifdef SKDOS
#define BINARY "rb"
#endif
#ifndef BINARY
#define BINARY "r"
#endif

FILE *file1;
FILE *file2;
char *arg;
char *prog;
int base;
int c1;
int c2;
int eflg;
int lflg = 1;
long chr;
long line = 1;
long skip1;
long skip2;
long v;

main(argc, argv)
int argc;
char **argv;
{

prog = argv[0];
if (argc < 3)

narg(prog);

arg = argv[1];
if ((arg[0] == '-') && (arg[1] == 's'))

lflg--, argv++, argc--;

arg = argv[1];
if ((arg[0] == '-') && (arg[1] == 'l'))

lflg++, argv++, argc--;

if (argc != 3)

narg(prog);

if (!(file1 = fopen(arg = argv[1], BINARY)))

barg(prog);

if (!(file2 = fopen(arg = argv[2], BINARY)))

barg(prog);

if (argc > 3)

skip1 = otoi(argv[3]);

if (argc > 4)

skip2 = otoi(argv[4]);

while (skip1)
{

if ((c1 = getc(file1)) == EOF)

arg = argv[1], earg(prog);
skip1--;

}
while (skip2)
{

if ((c2 = getc(file2)) == EOF)

arg = argv[2], earg(prog);

skip2--;

}
for ( ; ; )
{

chr++;
if ((c1 = getc(file1)) == (c2 = getc(file2)))
{

if (c1 == '\n')

line++;

if (c1 == EOF)

exit(!!eflg);

}
else
{

if (!lflg)
```

```
exit(1);

if (c1 == EOF)

arg = argv[1], earg(prog);

if (c2 == EOF)

earg(prog);

if (lflg == 1)
{

printf("%s %s differ: char %ld, line %ld\n",

argv[1], arg, chr, line);

exit(1);

}
eflg = 1;
printf("%6ld %3o %3o\n", chr, c1, c2);

}

}

}

otoi(s)
char *s;
{

base = 10 - ((*s == '0') << 1);
for (v = 0; isdigit(*s);

v = (v * base) + (*s++ - '0'));

return (v);

}

narg(s)
chsr *s;
{

fprintf(stderr, "usage: %s [-1] [-s] file1", s);
fprintf(stderr, " file2 [skip1] [skip2]\n");

exit(2);

}

barg(s)
char *s;
{

if (lflg)

fprintf(stderr, "%s: can't open %s\n", s, arg);

exit(2);

}

earg(s)
char *s;
{

fprintf(stderr, "%s: EOF on %s\n", s, arg);
exit(1);

}
```

+++

ARE UNIX AND OS-9 COMPATIBLE? SHELL, YES!

By : *Glenn Wood, Senior Programmer*
Ultrascience
A Division of Gibbs Laboratories, Inc.
Wilmette, Il

The rate at which the Bourne shell (Steven R. Bourne) has gained in popularity and migrated from UNIX, where it is the "standard" shell, to other operating systems testifies to its usefulness. Unlike many shells, which simply execute commands sequentially, the Bourne shell is a *complete* command programming language. It provides the sophistication necessary for linking complex and diverse programs, perhaps written in different languages, into smoothly integrated software systems with features not necessarily present in the individual components. Yet, the basics are so easily mastered that it makes an extremely friendly programming interface for any computer. Ultrascience offers the Bourne shell as a significant enhancement to the already rich command set of the OS-9 operating system. The Ultrascience implementation of the Bourne shell for OS-9 is called the "S/R Shell". Since both the Bourne shell and the S/R Shell have the same features, I will refer to them together simply as the "*shell*".

Programmers, familiar with shells, may ask why Ultrascience chose the *shell* and not the slightly more advanced "Korn shell" (David Korn) or "C shell" (Bill Joy). C shell is known for its interactive features and the Korn shell combines the interactive features of the C shell with greater efficiency. The decision was based on popularity. Because so many more people know and use the *shell*, it represents a better value. The larger market base translates to better support, greater availability of applications and qualified programmers, and lower purchase price. Most users will find the *shell* to be one of the most useful and cost-effective pieces of software they can install on their systems.

...one of the most useful and cost-effective pieces of software they can install on their systems.

I can't do justice to the power of the *shell* in a few pages; however, a brief review of the highlights should be of interest to anyone who is not already familiar with it. There are many excellent texts which describe the use of the *shell* in detail[1,2,3,4] and technical articles abound. The main features of the *shell* can be grouped as follows:

- **Script Files and Parameters**
- **Variables**
- **Block-structured Command Language**
- **Command substitution**
- **Wild-carding**
- **Piping and redirection**
- **Miscellaneous**

SCRIPT FILES AND PARAMETERS

The OS-9 command to invoke the *shell* is "srshell". It can be invoked to execute a single shell command then exit back to the OS-9 shell when it has completed its specified task, "srshell" task", or it can be started so that it remains active, "srshell", executing all the user's commands until the user quits with an escape. When the *shell* remains active, it will process sequences of shell statements entered from a user's keyboard, or it can process previously created "script files". As their name implies, script files are scripts for a sequence of shell statements, and they can be created using any text editor. Any OS-9 command or shell command (see Table 1) can be used from within the *shell*. Since the *shell* is a superset of the OS-9 shell, it can be installed in place of the OS-9 shell.

Table 1 - Shell Commands

:
No effect: the command does nothing and returns zero exit status.

.file
Read and execute commands from file and return. The search path $PATH is used to find the directory containing file.

break [n]
Exit from the enclosing for or while loop, if any. If n is specified then break n levels.

continue [n]
Resume the next iteration of the enclosing for or while loop. If n is specified then resume at the nth enclosing loop.

cd [arg] Change the current directory to arg. The shell parameter $HOME is the default arg. Also, the shell variable CDPATH is used to search for directories containing arg. This is fully described in section 4.1.4.

echo [arg...]
Echo arguments. See the echo command.

eval [arg...]
The arguments are read as input to the shell and the resulting command(s) executed.

exit [n]
Causes the shell to exit with exit status n. If n is omitted then the exit status is that of the last command executed. (An end-of-file will also exit from the shell.)

export [variable...]
The given variables are marked for automatic export to the environment of subsequently executed commands. If no arguments are given then the exported variable names are listed.

read [variable...]
One line is read from the standard input: successive words of the input are assigned to the variables in order; any remaining words are assigned to the last variable. The return code is 0 unless an end-of-file is encountered.

readonly[variable...]
The value of the given variables may not be changed by subsequent assignment. If no arguments are given then a list of all readonly variables is printed.

return [n]
Causes a function to exit with the return value specified by n. If n is omitted, the return status is that of the last command executed.

set [-aefhkntuvx[arg...]]

-a	Mark variables which are modified or created for export.
-e	If non-interactive then exit immediately if a command fails.
-f	Disable file name generation.
-h	Locate and remember function commands as functions are defined.
-k	All keyword arguments are placed in the environment for a command, not only those that precede the command name.
-n	Read commands but do not execute them.
-t	Exit after reading and executing one command.
-u	Treat unset variables as an error when substituting.
-v	Print shell input lines as they are read.
-x	Print commands and their arguments as they are read.
-	Turn off the -x and -v options.

Using + rather than - causes these flags to be turned off. These flags can also be used on invocation of the shell. The setting of flags may be found in $-. Remaining arguments to the set command are assigned, in order, to $1, $2,... If no arguments are given then the values of all names are printed.

Table 1 continued on next page

Script files can accept parameters, just as any program would. One class of parameters, called "positional parameters", are passed by listing them after the name of the script file.

myjob ***abc lm wxyz ... ..***

Within the script file, positional parameters are designated as $1, $2, $3, ... (see Table 2). The script positional parameters will be replaced by the arguments in the order they were issued: $1 is replaced by "abc", $2 is replaced by "lm", etc. $0 is a special parameter which is replaced by the name of the script file, "myjob".

Positional parameters, excluding $0, can also be set by using the *set* command from within a script file. The output from *set* will be fed, sequentially, into the positional paramters until they are used up. Thus, "set *" (see Table 1) will replace $1 with the first file name in a directory, $2 with the second file name, and so on.

VARIABLES

The parameters concept is extended in the *shell* to a general purpose system of variables (see Table 2). Any word preceeded by a $ is interpreted by the *shell* as a variable and the current value of that variable is substituted in its place. The value of a variable may be set by a simple assignment statement within a shell sequence, or it may be passed to a script file as a parameter by equating it on the script command line, prior to the script name.

name=Glenn myjob ***abc lm wxyz ...***

The *shell* will replace $name with *Glenn* whenever it occurs, unless the value of the variable is changed as the *shell* sequence precedes.

Variables are very useful for tracking and controlling the operation of the computer. In the *shell*, variables may be equated by literal assignment, by parameter assignment, by default assignment, or as arithmetic or string calculations based on other variables (see Table 4).

Table 1 - Shell Commands (cont.)

shift [n]
The positional parameters from $n+1... are renamed $1... If n is not given, it is assumed to be 1.

test
Evaluate conditional expressions. See the test command.

trap [arg][n]...
arg is a command to be read and executed when the shell receives signal(s) n. arg is evaluated once when the trap is set and once when the trap is taken. Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal]](memory fault) produces an error. If arg is absent then all trap(s) n are reset to their original values. If arg is the null string then this signal is ignored by the shell and by invoked commands. If n is 0 then the command arg is executed on exit from the shell. Trap with no arguments prints a list of commands associated with each signal number.

unset [variable...]
For each variable, remove the corresponding variable or function. The variables PATH, PS1, PS2, and IFS cannot be unset.

wait [n]
Wait for the specified process and report its termination status. If n is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

Table 2 - Shell Variables

$?	The exit status (return code) of the last command executed as a decimal string.
$0	The name of the command procedure being executed.
$#	The number of positional parameters (in decimal).
$$	The process number of this shell (in decimal).
$!	The process number of the last process run in the background (in decimal).
$-	The current shell flags, such as -x and -v.
$HOME	The default argument for the cd command.
$CDPATH	The list of directories that is searched by the cd command.
$PATH	A list of directories that contain commands (the search path).
$PS1	The primary shell prompt string, by default, $.
$PS2	The shell prompts with $PS2 when further input is needed, by default, >.
$IFS	The set of characters used for blank interpretation.

- **name=Glenn**
- **bestname=$name**
  nextparm=$2
- **bestname=${name-$1}**
- **total='expr $total + $amount'**

The "command substitution" feature (see below) even allows the standard output text of a program to be assigned to a variable.

Variables can be *tested* and *compared* (see Table 5).

- **if [$total -ge 100] then . . . else . . . fi**
- **if [$bestname != ""] then . . . elif . . . fi**
- **while [$count -ne 0] do . . . done**

Variables also can be passed as a parameter to programs or other script files.

- **echo $bestname**
- **dir $1 $2 $3**
- **compile $progl $prog2**

BLOCK STRUCTURED COMMAND LANGUAGE

The *shell* command language is "block-structured" which means that the commands are composed of blocks of statements. Any block of statements can be considered as a single statement, so blocks can be composed of smaller blocks. Each block is structured in one of the following forms:

- **A simple command**
- **FOR..DO..DONE**
- **FOR..IN..DO..DONE**
- **WHILE..DO..DONE**
- **UNTIL..DO..DONE**
- **CASE..case1..case2..ESAC**
- **IF..THEN..ELSE..FI**
- **IF..THEN..ELIF..FI**

These programming constructs can be controlled by means of the results of programs (their 'exit' codes), the values of variables and parameters, and the attributes and ages of files. In fact, by utilizing "command substitution" (see below), the standard output text of a program can be used to direct subsequent operations!

WILDCARDING

Wildcarding is a familiar part of OS-9. It allows the naming of several files by means of a single wild-card specification. However, the *shell* provides more powerful wildcarding features, such as single character matches to a range of characters and the selection of files that do <u>NOT</u> match the wild-card specification.

- cat* — **any file name that starts with "cat"**
- [abc]xyz — **any file name beginning with "a", "b" or "c" (upper or lower case) and ending in "xyz".**
- [0-9]* — **any file name beginning with a digit.**
- [~0-9]* — **any file name NOT beginning with a digit.**

Table 3
Special Shell Characters

- `*` wildcard match to string.
- `?` wildcard match to character.
- `[]` wildcard match to any one of a set of characters.
- `[-]` wildcard match to sequences of characters.
- `&` places commands in background mode, leaving the terminal free for other tasks
- `;` separates multiple commands on one command line for sequential processing
- `()` groups commands for execution
- `\` turns off the meaning of special characters such as *, ?, [], &, ;, >, <, and |.
- `'...'` single quotes turn off the delimiting meaning of a space and the special meaning of all special characters
- `"..."` double quotes turn off the delimiting meaning of a space and the special meaning of all special characters except $ and `
- `1>` or `>` redirects output of a command into a file (replaces existing contents)
- `2>` redirects error output of a command into a file (replaces existing contents)
- `<` redirects input for a command to come from a file
- `>>` redirects output of a command to be added to the end of an existing file
- `<<` here document
- `|` creates a pipe of the output of one command to the input of another command
- `` `...` `` grave accents allow the output of a command to be used directly as arguments on a command line
- `$` used with positional parameters and user-defined variables
- `#` comment follows a command to be added to the end of an existing file

COMMAND SUBSTITUTION

I have already mentioned command substitution several times. It permits the standard output from any command to be used as input to shell sequence variables. A command, or string of commands, placed between grave accents (backquotes) causes the command output to be treated just as if it were a variable. In the following example, the output of the *setime* program is assigned to the "time" variable, so that the *echo* command would issue "October 9, 1989 Monday 8:05:11 pm".

```
time=`setime -s`
echo $time
```

PIPING AND REDIRECTION

The *shell* provides all the conventional piping and redirection features present in OS-9, so that program input and output can be diverted to or from standard I/O or between programs (see Table 3).

MISCELLANEOUS

The *shell* provides for error and fault handling (see "trap" Table 1), as well as the debugging of script files (see "set -v" and "set -x" Table 1).

PULLING IT ALL TOGETHER

The following is a useful illustration of how a number of the *shell* features tie together into a practical, powerful programming tool. It is a simple script file that compiles C programs, sending the compile report to a disk file, and then, only if each compilation was successful, links the programs together.

```
# SCRIPT FILE 'ccc'
 failed=0
 for source
   do
          if cc ${source}.c -sr
            then
              failed=1
          fi
   done
 if [ $failed -eq 0 ]
   then
          link_prog
   fi
# end of script 'ccc'
```

My programs are usually made up of more than one module. In the course of debugging them, I may change several modules at once, then compile and link them together. This script file allows this to be done with a single command:

```
ccc f1 f2 f3 >err
```

By redirecting the output from the script file with the ">err" term, any compilation errors will be placed in a file named "err", which I can read later. I could add an *&* to the end of this command, which would put the whole script file execution into background. Then, I could continue work from my terminal on some other problem while the "ccc" script is being executed.

In this sample script file, the "for" command causes each module name (f1, f2 and f3) to be assigned in turn to the variable "source". The "if cc ....." command line compiles each source and tests the result at the same time. If any one of the compilations fails, "failed" is set to 1; no computer time, or programmer time, is wasted linking bad modules together. If all compilations succeed, then another script file, "link_prog", will be executed to link all modules together.

Someone may be thinking, "the OS-9 *cc* compiler can compile several programs at once, why bother." The OS-9 *cc* compiler will abort all subsequent compilations if any one fails; I prefer to see all of my errors at once, and fix them all at once. By composing this simple script file, I can make the *cc* compiler behave the way I want it to, simplifying my work and increasing my productivity. I use the *shell* to perform all manner of housekeeping tasks and preparation of serialized software releases; life before the *shell* was ....., a lot harder.

[1] Stephen R. Bourne, *THE UNIX SYSTEM V ENVIRONMENT*, Menlo Park, CA, Addison Wesley Publishing Company, 1987 (ISBN 0-201-18484-1)

[2] Mark G. Sobell, *A PRACTICAL GUIDE TO UNIX SYSTEM V*, Menlo Park, CA, The Benjamin/Cummings Publishing Company, INC., 1985 (ISBN 0-8053-8915-6)

[3] AT&T, *UNIX SYSTEM V, Release 3.2*, Englewood Cliffs, NJ, Prentice-Hall, Inc., 1989 (ISBN 0-13-944133-6)

[4] Kaare Christian, *THE UNIX OPERATING SYSTEM*, New York, NY, John Wiley & Sons, 1988 (ISBN 0-471-84781-X)

Table 4

expr - evaluate arguments as an expression

expr arg ...

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Note that 0 is returned to indicate a zero value, rather than a null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr | expr

Return the first expr if it is neither null nor 0; otherwise return the second expr.

expr & expr

Retun the first expr if neither expr is null or 0; otherwise return 0.

expr {=, >, >=, <, <=, !=} expr

Return the result of an integer comparison if both arguments are integers; otherwise return the result of a lexical comparison.

expr {+, -} expr

Addition or subtraction of integer-valued arguments.

expr {*, /, %} expr

Multiplication, division, or remainder of the integer-valued arguments.

expr : expr

The matching operator : compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of ed(1), except that all patterns are anchored at the beginning of the string. Therefore, ^ is not a special character in that context. Normally, the matching operator returns the number of characters matched (0 on failure). Alternatively, the (...) pattern symbols can be used to return a portion of the first argument.

Table 5

test - condition evaluation command

test expr

Test evaluates the expression expr and, if its value is true, returns a zero exit status; otherwise, a non-zero (false) exit status is returned; test also returns a non-zero exit status if there are no arguments. The following primitives are used to construct expr:

-r file	True if file exists and is readable.
-w file	True if file exists and is writable.
-x file	True if file exists and is executable.
-f file	True if file exists and is a regular file.
-d file	True if file exists and is a directory.
-p file	True if file exists and is a named pipe (fifo).
-k file	True if file exists and its sticky bit is set.
-s file	True if file exists and has a size greater than zero.
-t [fildes]	True if the open file whose file descriptor number is fildes (1 by default) is associated with a terminal device.
-z s1	True if the length of the string s1 is zero.
-n s1	True if the length of the string s1 is non-zero.
s1 = s2	True if strings s1 and s2 are identical.
s1 != s2	True if strings s1 and s2 are not identical.
s1	True if s1 is not the null string.
n1 -eq n2	True if the integers n1 and n2 are algebraically equal. Any of the comparisons -ne, -gt, -ge, -lt, and -le may be used in place of -eq.

These primaries may be combined with the following operators:

!	Not.
-a	And.
-0	Or.
(expr)	Parentheses are for grouping.

All the operators and flags are separate arguments to test.

UNIX is a trademark of AT&T Bell Laboratories

OS-9 is a trademark of Microware Systems Corp.

S/R is a trademark of Ultrascience, a division of Gibbs Laboratories, Inc.

GRAB YOUR OS-9 APPLICATIONS MANUALS

By : *Jan Johansen, President*
Johansen & Associates
Park Ridge, IL. 60068

OS-9 does not want for praise, but it has not yet been equipped with sophisticated desktop publishing tools like Ventura or PageMaker, or even simple page layout features like those found in Word Perfect 5.0 or Microsoft Word. When an OS-9 applications programmer wants a professional looking manual, we take the ASCII output from his OS-9 word processing programs and feed it into personal computer desktop publishing software. The results have been pleasing and, judging from our repeat business, effective. We have resources for translating text into all the major languages, and we employ graphic design experts who know how to create dramatic formats and use a variety of graphics packages to embellish a document with exciting "clip-art". However, until recently, our style was a bit cramped by the fact that we had no convenient way to create screen-grabs from our clients' OS-9 applications.

Anyone who has ever used an applications manual knows just how helpful the image of an actual application screen can be. A picture is worth a thousand, otherwise possibly tedious, words. In our experience, it takes much less time to write a software manual which contains screen-grabs, and the associated products are much easier to sell.

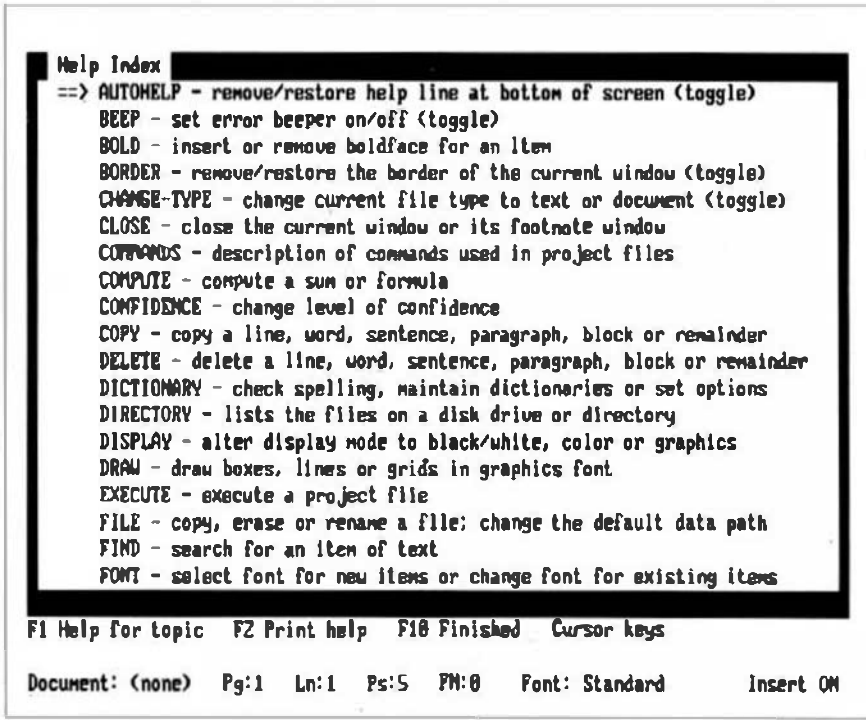
Anyone who has ever used an applications manual knows just how helpful an image of the actual screen can be.

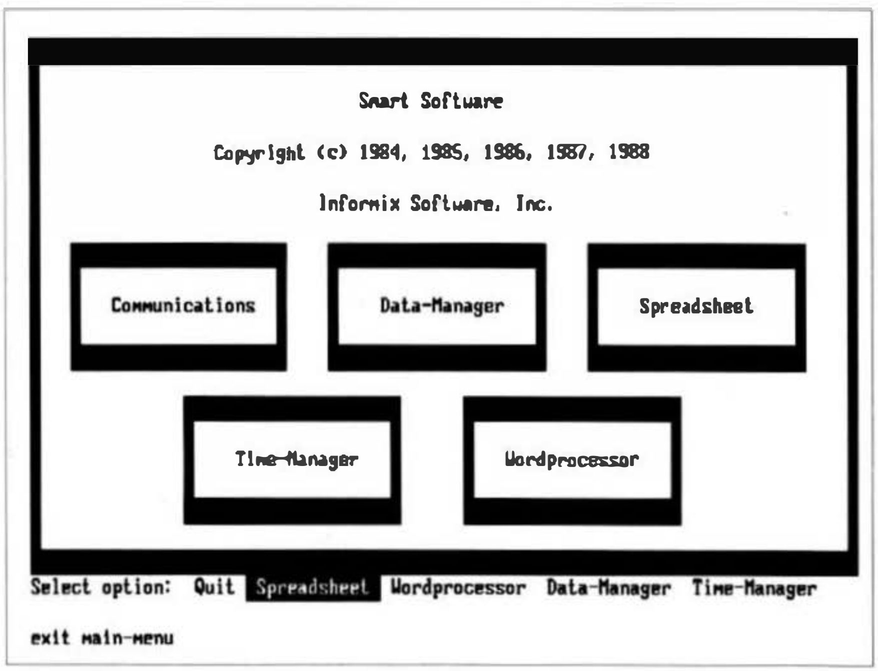
Now there is a solution! Ultrascience has created a 68K coprocessor board, the PC68K1, which runs OS-9 inside virtually any IBM-style personal computer[1]. The board was designed to provide a cost-effective multi-user environment for personal computers, but we are recommending it to our OS-9 based clients as a low-cost way to obtain screen-grabs for their manuals. The cost savings on the first manual usually washes out the cost of the PC68K1 (suggested list under $2000.00).

The PC68K1 and OS-9 install in minutes. OS-9 runs concurrent with DOS and the monitor can be hot-keyed back and forth between the two operating systems. First, we boot OS-9 in memory resident mode. Then, we start HOTSHOT (this sequence is important in order to avoid an error). HOTSHOT (suggested list price $249.00[2]) permits the user to select a "hot key" to save the image of any monitor screen to disk. As an OS-9 application runs, the operator simply enters the screen-grab hot key when he wants to record a screen. HOTSHOT will prompt for a path and file name, and the screen is recorded to disk - it's that simple.

Using Ventura, we can blend our clients' OS-9 screen-grabs into their manuals. We can also edit the screen-grabs, maintaining the appearance of almost any font or graphics. This editing ability is particularly useful for "blanking" serial numbers or combining the effect of sequencing through several fields on a screen into a single grab.

The combination of running OS-9 on a PC and using HOT-SHOT to produce and edit valuable screen visuals has certainly streamlined our documentation process and, more importantly, has made the final result much more effective. If you have any questions, please feel free to call or write us at 910 Busse Hwy., Park Ridge, IL 60068, phone: 312/698-9471, fax: 312/825-2737.





NOTE: We had already submitted this technical note when we learned that Ultrascience is providing OS-9 for Macintosh computers. It seems likely that the COMMAND-SHIFT-3 feature of MAC software may create a useable MAC .pic file from OS-9 screens. We will report on this possibility in a future issue.

[1] Ultrascience
A Division of Gibbs Laboratories, Inc.
1824 Wilmette Avenue
Wilmette, IL 60091

[2] HOTSHOT Graphics
SymSoft, Inc.
P.O. Box 4477
Mountain View, CA 94040

South East Media

Telephone: (615) 842-4600

OS-9, UniFLEX, FLEX, SK*DOS Fax: (615) 842-7990

SOFTWARE

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only. FLEX, SK-DOS & CCF - $49.95
Disk Set (2) - FLEX, SK-DOS & CCF & OS-9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

FLEX, SK-DOS $129.50
SPECIAL PAT/JUST COMBO (with source)
FLEX, SK-DOS $99.95, OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

FLEX, SK-DOS $69.95

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc. Complete with over 25 different CRT terminal configuration overlays.

FLEX, CCF, SK-DOS $39.95

SPELLB "Computer Dictionary" from S.E. Media -- **OVER 150,000 words!** *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX, SK-DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

FLEX, SK-DOS and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95,
FLEX, SK-DOS or OS-9 - $179.95, UniFLEX- $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - $69.95,
FLEX, SK-DOS or OS-9 - $99.95, UniFLEX- $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES 6809 CCF and CCO - $59.95,
FLEX, SK-DOS or OS-9 - $79.95, UniFLEX- $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!

FLEX, SK-DOS or OS-9 - $329.95, UniFLEX - $549.95
OS-9 68000 $695.00

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems
FOR 6809 FLEX or SK-DOS (5/8")

Up to 32 groups/fields per record! Up to 12 character file names! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software sys ms with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX or SK-DOS(5"/8" Disk) $249.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343
Telephone: (615) 842-4600 FAX (615) 842-7990

mastercard

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

OS9
"File Manager Round Up"

"Steven Weller Windsor Systems"

File Manager Round Up

The key to fast, effective and convenient use of OS-9 I/O lies in the choice of file manager. Use the wrong one and the result is likely to be slow, inefficient, or just plain awkward to use, however good the driver. Coding around the problems can help, but at the expense of reduced portability and increased programming time. There is more to life than SCF and RBF for controlling I/O devices, as this article explains.

This article briefly describes an array of OS-9/68k file managers that are available to reach the parts that the Microware file managers cannot. It goes on to consider the pros and cons of growing your own file manager at home, giving some idea of the strategy needed and problems you are likely to encounter.

Microware's standard file managers cater well for a wide range of standard devices including terminals, magnetic disks, tapes, and networking and provide an obvious first choice for most applications. Other classes of devices, and particulary intelligent devices, need file managers written specifically to match their capabilities. Analog to digital converters are simple devices to operate, for instance, and may be driven through SCF, but become severely speed-limited by the overheads that SCF imposes. Intelligent devices cause more problems since they are almost always block-oriented and multi-channel - SCF is neither.

There is a surprisingly rich choice of OS-9 file managers currently available. Off the top of my head I can think of fourteen: seven from Microware. They cover a wide range of applications, including IEEE-488 devices, graphics, WORM disks, intelligent I/O cards, pipes, and PC disks.

Here is a run-down of the third-party file managers I know of (for suppliers see the end of this article):

SPF
Sequential Packet
File Manager

Galactic Industrial's SPF is a block-oriented multi-channel file manager aimed at intelligent I/O boards. All file manager entry points are passed directly to the driver, allowing intelligent serial, ADC, DAC, and signal processing cards to be driven at very high speed. Read and write accesses are separately queued, and can be run independently. Additionally, it has the line editing capabilities of SCF, so can be used to drive normal terminals through intelligent serial boards. Source for an example driver is supplied.

Gksman
Graphics File Manager

Snowtop Computer's Gksman is an escape-sequence driven multi-channel graphics file manager. It performs a wide range of graphical functions including windowing, clipping, polylines, markers, high-speed text, segments, input echo, mouse control and pointer display, wide lines, complex polygon filling and cross-hatching, and can emulate SCF to provide a terminal on a graphics screen. It can drive graphics chips, bitmapped displays, intelligent workstations, dot-matrix and laser printers, and pen plotters. It is supported by C libraries and includes an input event queueing system.

MFM
Memory File Manger

The Memory File Manager from Windsor Systems operates much like a RAM disk under RBF with several important differences. It dynamically sizes the memory "disk" using only as much system memory as it requires to store the files it has. Aimed at ROM-based applications, it supports only a single directory level for simplicity and speed, and can access data modules as files, allowing files like 'startup', 'termcap', and 'umacs.hlp' to be ROMmed for use by any program. The equivalent of the sector size in MFM is user-selectable and MFM can replace RBF in many ROM-based applications.

WRF
WORM File Manager

The Write Once Read Many optical disk file manager from Galactic Industrial solves many of the problems associated with the relatively slow WORM media. It supports the hierarchical RBF directory structure, caching directory entries to reduce seek times. It allows multiple files to be open for reading, but only one file to be open for writing. Files can be modified and deleted, and all old versions of files are accessable. In addition, logical interleaving and file storage in sequential logical sectors improves the access speed.

MSF
MS-DOS Disk File Manager

Galactic Industrial's MSF goes much further than Microware's PCF PC disk file manager. It allows all utilities but dcheck, format, and free to be used on MS-DOS formatted disks and provides three extra utilities to replace them. File names and data structures are automatically converted between the two systems, and all formats from 360K to 1.44M (and hard disks) are supported. Additionally, a utility is provided to convert existing descriptors to MS-DOS descriptors. Many existing drivers and applications programs will work with MSF without modification.

IBF
IEEE-488 Bus File Manager

ARK Corporations's IBF provides multi-channel control of IEEE-488 bus equipment. It automatically handles all addressing protocols, allowing IEEE-488 devices to be simply driven with normal I/O system calls. Fifteen debugging utilities are provided, allowing devices to be controlled from the command line. A C library gives access to all facilities via GetStatus and SetStatus calls. It is block-oriented, and the drivers supplied support DMA operation. A locking mechanism prevents collisions from multiple processes.

PC9
PC to OS-9 File Manager

Quin Systems' PC9 system allows an MS-DOS computer to be used as a terminal to multiple processes on a remote OS-9 system linked by a single serial cable. Each OS-9 process displays through a resizable, moveable window on the PC screen. Terminal emulation facilities support uMACS and other screen editors and provide a programmable PC keyboard. Access to PC disk drives is also available through the OS-9 unified I/O system, giving disk capability to ROM based OS-9 systems. A hot key switches between DOS and OS-9 displays. A fast version driving Quin's Topaz LAN will be available Q2 90.

Logio
Logical I/O File Manager

Windsor Systems' Logio is a multi-channel block-oriented file manager controlling non-intelligent digital and analog I/O devices. It enables physical devices to be split into a number of logical devices for use by multiple processes and matches physical data size and alignment to that used by the calling program. Signals and events can be triggered when physical conditions become true, allowing alarm condition monitoring. Future developments include automatic linearization for analog devices and facilities for accurately timed input and output. Logio will be available Q1 90.

Do It Yourself File Managers

Writing your own file manager can be a very rewarding, if costly, experience. The average licence price of the file managers just described is around $1900, somewhat less than the cost of a programmer for a week. The end result, however, can fit your application exactly.

To write an OS-9 driver you must know something of the internal workings of OS-9, have the technical documentation for the appropriate file manager to hand, and understand the hardware of the device you wish to drive. To write a file manager is a much harder job, not because the code is any more difficult, but because a great deal more knowledge and experience os OS-9 is required.

Since the file manager interfaces the kernel to the driver, the kernel-file manager interface has to be well understood in order that the file manager can be written at all. The file manager-driver interface, the path descriptor, device descriptor, and static storage definitions are all defined by the file manager writer and must be carefully thought out. As the file manager designer, this freedom comes at a price: you are unfortunately also expected to write a great deal of technical documentation for those who will write drivers for it.

The freedom that a home-grown file manager gives is enormous. For instance, the file manager controls simultaneous accesses by multiple processes on different paths (the kernel queues multiple accesses on the same path), so the file manager writer has complete control over multi-process access to the driver, possibly, like SPF, giving all the control to the driver. Processes can be created by a file manager (SBF and NFM do this), and data modules and other structures can be used to overflow the standard path descriptors and process descriptors. File managers can be written to emulate SCF in order to include features that are not currently supplied, such as multi-channel access or improved line editing facilities, as has been done in PC9. Either C or assembler can be used to write file managers; my choice is C. Once the interfaces to the driver and kernel are written in assembler, the internals are most easily coded in C. Using C for drivers is a little more tricky because of the high degree of hardware interaction and the need for speed. OS-9000, written almost entirely in C, almost requires that file managers are coded likewise. For OS-9000 the kernel and driver interfaces have been made C-friendly and file manager static storage added, solving some of the problems that the next section deals with that exist on OS-9/68K.

What is the downside of writing your own file manager ? There are four problems that cause the most aggravation.

The first is the very large amount of information that must be considered before coding begins. Start by designing the data structures and define how simultaneous accesses to drivers are to be arbitrated. Then consider the needs of the driver and its operation, and start coding Open, Create, and Close first. It is useful to create a 'null' driver at this point and test what you have. Debugging can be quite effectively carried out via Microware's system state or ROM debuggers, or through messages output to a terminal.

The second problem is concerned with overflowing data structures. Both the path descriptor and the process descriptor overflow quite quickly. The path descriptor overflows simply due to the amount of path-specific data needed to be stored by complex file managers. Storing a pointer to extra storage allocated from system memory in the path descriptor takes care of that. The process descriptor stores the process system stack and C programs arevery stack-hungry. An overflowing system stack usually manifests itself in the form of weird errors and random system crashes. The solution is to maintain a pool of large stack buffers and transfer the stack on entry to and exit from the file manager.

The third problem involves memory management and error handling. In system state (your file manager code is running in system state) memory allocation is attributed to 'the system', not to any individual process. Any memory not returned to the system is lost forever ! Error handling, unless very carefully thought out, can result in lost memory. You have to write your own memory allocation routines, and it is worth having a method of logging all memory allocated by the file manager (via a linked list) in the

path descriptor, so that when a path is closed an error can be generated, and/or the 'lost' memory can be returned to the system.

The fourth problem is that static storage and initialised data are not allowed. This has three implications. The first implication is that many of the standard C library routines use initialised or static data and therefore cannot be used. Any ROF that contains a function with such needs makes the whole ROF unusable. The only way around the problem is write your own library routines. The second is that your code cannot contain some constant arrays, such as arrays of pointers. The third is that the jump table (automatically created by the linker for branches over 32K) cannot be used. The various parts of a large file manager have to be carefully arranged, or some alternative calling technique employed (an array of pointers to functions for instance).

Once you have written your first file manager, the second one comes much more easily. You have the assembly language interface and library of routines for accessing system calls written, and you have the experience behind you. You do, of course, still have a pile of documentation to complete....

File Manager Suppliers

In North America contact:
Steven Weller at Windsor Systems (U.S.A) +1 (502) 425 9560 (voice) +1 (502) 426 3944 (fax) for file manager and driver writing services, MFM, SPF, WRF, MSF, IBF, PC9, Logio and Gksman.

In Europe contact:
Paul Dayan at Galactic Industrial (England) +44 (582) 405759 (voice) +44 (582) 450324 (fax) for file manager and driver writing service, SPF, WRF, MSF, MFM and IBF.
John Poulter at Snowtop Computers (England) +44 (582) 451084 (voice) +44 (582) 20764 (fax) for Gksman, IBF and MFM.

John Withers at Quin Systems (England) +44 (734) 771077 (voice) +44 (734) 776728 (fax) for PC9.

In Japan contact:
Toshikuni Osogoe at Osque Systems (Japan) +81 (3) 220 0384 (voice) +81 (3) 220 0417 (fax) for IBF and MFM.

+++

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01453

GETTING STARTED IN FORTH

While the proliferation of versions of Forth has the advantage of supplying something for just about everyone, it does leave a lot of confusion in the mind of a beginner. It is possible to find a Forth which will run on just about any machine with just about any operating system. In most cases, any Forth you might choose would be satisfactory; therefore, I don't see that there is much room for argument, here. Under these circumstances, my recommendation is that you get the Forth which fits your favorite operating system and start having fun!

Some of the possibilities are listed in Table 1. This is not meant to be an exhaustive list by any means, but it does list those Forth versions that I have tried personally.

Once you get your Forth, you will probably be swamped by the several hundred words at your disposal in the various vocabularies. Don't be intimidated by what looks like a lot of memorizing for you to do. Many of these word are used only for special cases which you may never encounter; there are words which I have never used and never felt the need for, so just relax and "go with the flow."

You will find that you will need about 45 words immediately. The rest can be learned as you need them. This minimum list can be found in Table 2.

I am sure that I have left out some of the favorite words of those of you who have been writing Forth programs for a while, but remember that this is a list of the MINIMUM requirement, and not any particular list of favorites. This list was generated with the help of the local FIG chapter; we spent a whole meeting session working on it. Particularly, I want to thank Dick Miller of Miller Microcomputer Services for helping me get started with the list.

The categories within the list have been arranged in alphabetical order and there is some grouping within each category, but I doubt that any group of words within the list is really more important than another, unless you have special programming requirements.

I will discuss the categories in the order in which they fall in Table 2.

Arithmetic words:

Notice, first, that these are all words directly related to 16-bit integers. We felt that the beginner would have little or no need for 32-bit arithmetic, so those words could be learned later.

The use of the first four words, + - * and / , should be obvious to anyone who has ever written even the simplest program in any language; however, the others may not be as easy to grasp.

MOD is the remainder after an integer division and is usually lost if you don't make a special provision to retain it. Remember that in integer division 6/3, 7/3, and 8/3 all have the save quotient of 2, so you must make provision for saving the remainder if you need it. MOD saves only the remainder from the division, while / MOD (pronounced "slash mod") saves both the quotient and the remainder.

*/MOD is a special case of a combined multiplication and division operation which makes use of a 31-bit intermediate product. This helps maintain a greater accuracy in the results, but you have no access to the intermediate product. If you don't know when you will need this word, then you probably don't need it!

NEGATE and ABS are not complementary just because they operate on the sign of a number; therefore, don't be too complacent with them. NEGATE leaves the two's-complement of a number, which means that you could be surprised at the result of NEGATE operating on a number which is already negative. On the other hand, ABS simply takes a signed number and makes it act as an unsigned number, which is not necessarily the same as converting a negative number into a positive number. In other words, these two operations can bite the hand of the unwary!

MAX and MIN , which compare the top two 16-bit numbers on the Data Stack and select the higher magnitude or the lower magnitude, respectively; the other value disappears from the stack. I think that it is a valid assumption that, if the two numbers are the same, then one is simply erased. The action of MAX and MIN is relatively obvious when you realize that each word acts on a pair of signed numbers to return the MAXimum or MINimum of the pair.

Comparison words:

These are the simple "less than", "equal to", or "greater than" functions for determining the relative magnitude of a pair of signed numbers. In contrast to MAX and MIN , < , = , and > return a boolean flag replacing both numbers, so you better be sure to save them either on the Data Stack or in variables before you make the comparison if you expect to have any further use for either one.

Control structure words:

Forth has a number of additional control structures, but we felt that the ones in the list could be used, in combination if necessary, to satisfy any common requirement.

Consider the possible forms a control structure can take:

|-| 1. Either/or controlled by IF ... ELSE ... THEN
|-| 2. Definite loop controlled by DO ... LOOP
|-| 3. Indefinite loop controlled by BEGIN ... UNTIL

What other types of decision could one make? Any other decisions must be variations on one of these three basic types.

Probably, the most commonly used control would be the IF ... THEN or the IF ... ELSE ... THEN form. In either case, the most accessible number on the Data Stack will be used as a boolean flag to control the flow of the program; this flag is lost in the test for true/false so it cannot be used later. A true condition (any number not equal to 0) causes the words after IF to be executed; once all of these words have been run, control jumps to the first word following THEN . If ELSE is present, a false condition (a 0) causes the words following ELSE to be executed. When the ELSE words have all been run, or if there is no ELSE phrase present, then operation jumps to the first word past THEN.

I found this to be confusing when I first started working with Forth, since I had been using FORTRAN and BASIC, but I had very little trouble remembering to set the true/false before the IF , rather than after it. Once I got that straight, I had no more trouble with this structure.

The other two control structures are examples of a definite and an indefinite loop. The DO ... LOOP is a definite structure in which you know how many times the loop will be executed before you ever enter it, but the BEGIN ... UNTIL is an indefinite structure because it is executed until a true boolean flag appears on the Data Stack; if the flag were always false, then the loop would run forever!

One of easiest things to forget when setting up a DO ... LOOP is the proper order for the numbers defining the loop limits. You must put the end-value first and the start-value second (after all, this is an RPN system!) or you will get a surprise! It still happens to me, sometimes, and I can usually laugh at the result, but not always. It is occasionally hard to remember what value to use for the end-value; simply add the desired number of iterations of the loop to the value of the start-value. You will get 10 iterations from 10 and 0, 5 and -5, or 123 and 113; the only significance to the actual magnitude

of the starting limit is whether or not you want to use the internal loop counter for calculations.

You can access the internal loop counter with the word I (pronounced "eye"). I always refers to the counter for the current loop, so be sure which counter you want when you are working with nested loops.
The default stepping rate for the DO ... LOOP is +1, so you must use a different form if you want a different rate. This form is DO ... n +LOOP , in which the "n" is the magnitude of the step. It can be any value, except 0, and have either a + or a - sign. If "n" is negative, be sure your start value and end value are in the proper order! Also, remember that you must use +LOOP even when "n" is a negative number.

As I said before, the BEGIN ... UNTIL loop can run forever if you don't pay attention to how you set it up. The indefinite loop is abandoned only when the Data Stack contains some value not equal to 0 just prior to execution of the UNTIL . This number acts as a flag, so it is consumed by the test done by UNTIL ; therefore, if you want to use it somewhere else, be sure to DUP it. You can achieve an infinite loop by writing a definition which ends ... 0 UNTIL ; . Since this forces the flag tested by UNTIL always to be FALSE , the looping will be infinite, so this is one of the common ways used to set up the outer shell of a program such as an editor, in which you want to keep repeating the main routine.

Data Stack words:

Of course, very nearly everything that you do in FORTH makes at least one manipulation of the Data Stack, so don't confuse that fact with the present discussion. What I have in mind here are just those words which do nothing but operate on the Data Stack.

DROP removes the top number from the Data Stack and DUP duplicates the top number on the stack. SWAP exchanges the top two numbers on the Data Stack. OVER can be a little confusing at first, but it can be thought of as a DUP somehow performed on the number next to the one which was on top of the stack.

ROT is something like SWAP , except that it works on the third integer on the Data Stack. This integer is simply moved into position so that it becomes the first number on the stack. The two integers which formerly were the first and second numbers now become the second and third, without their order being changed.

Each of these words affects the count of the numbers on the Data Stack in this way:

|-| DROP removes one integer from the Data Stack.
|-| DUP adds one integer to the Data Stack.
|-| OVER adds one integer to the Data Stack.
|-| ROT does not change the count of integers on the Data Stack.
|-| SWAP does not change the count of integers on the Data Stack.

Defining words:

This sounds rather formidable, since many FORTH books make a big deal about how easy it is for one to create new defining words which extend FORTH in a multitude of directions (all at once?). I think I understand enough about "defining words" to believe that most of them are of no use to the beginner, although an expert FORTH programmer delights in using them (I am not an expert!). At least 99% of my FORTH programming uses only four words which fall into this category, and they are listed in Table 2.

You must use the : to open a definition and the ; to close a definition. That is about it!

The only other words which most people use extensively are CONSTANT and VARIABLE , which are technically "defining words", but already exist in the FORTH vocabulary, so you don't have to worry about them.

If you don't understand how to write your own "defining words", don't worry about it; you may never even care! For now, skip those chapters and read them later, if you ever have the time or feel deprived.

Input word:

KEY is the only input word a beginner really needs to become comfortable with, even though there are other input words contained in the various standards.

KEY simply takes one character from the input stream, usually the keyboard, and puts it onto the Data Stack. Anything else that happens from then on is up to the programmer. As a result, KEY can be the foundation for building very elaborate input structures which should cover all of your needs.

Logic words:

Forth logic is usually at the bit level, so that AND and OR are always "bitwise" operations. You must remember that in order to avoid confusion. I won't spend time on AND and OR , since they are standard throughout Forth and "bitwise" operations are simple to understand.

Memory access words:

Most memory access is done as 8-bit or 16-bit units in Forth, although there are exceptions. The only important consideration which might not be obvious is that storage of an 8-bit value (C!) into memory is done from the B Accumulator and not the A Accumulator with the 6800 and 6809. People who are used to writing Assembly programs usually use the A Accumulator for most of their fetch and store operations, so they must keep this difference in mind with Forth.

An 8-bit fetch (C@) operation for the 6800/6809 places the value into the B Accumulator and clears the A Accumulator, thus immediately promoting the 8-bit value into a 16-bit value. This can also cause problems for Assembly programmers if they don't keep it in mind.

Misc. & System words:

WORDS , known as VLIST in Fig-Forth, is important, since it shows you the defined functions you already have at your disposal. This command is often used in conjunction with FORGET .

FORGET removes the indicated word AND ALL FOLLOWING WORDS from the dictionary, so that they are not available for use. Therefore, you should make a habit of checking with WORDS before using FORGET, so that you can avoid unpleasant surprises.

Output words:

Only four words should satisfy a beginners needs for output, including formatting hardcopy. Simply put, . (dot) prints the numerical value of the integer on top of the Data Stack, and ." (dot-quote) prints the string pointed to by the integer on top of the Data Stack. Actually, the action of each is a little more complicated than that, but that is what you need to know to get started.

CR just sends out an "end-of-line" signal, which is usually $0D, but might be something else if the DOS needs it.

EMIT sends out the ASCII equivalent of the integer on top of the Data Stack. For example, if $41 or (decimal 65) were on top of the Data Stack, EMIT would send out an A.

Other words:

I have not included Editor, Assembler, or Disk I/O words, since they are too dependent on the machine and the DOS that you are using. I assume that you either know how to use them, already, or else don't need them.

Table 1. The 6800/6809 Forths which I have successfully used.

Processor	Operating System	Source	Type
6800	none RAM disk	Forth Interest Group P O Box 8231 San Jose, CA 95155	Fig-Forth
6809	FLEX	Forth Interest Group P O Box 8231 San Jose, CA 95155	Fig-Forth
6809	CoCo tape or disk	Stearns Electronics c/o South East Media 5900 Cassandra Smith Rd. Hixson, TN 37343	Fig-Forth with Forth-79 additions
6809	FLEX	Wilson M. Federici 1208 NW Grant Corvallis, OR 97330	Forth-83
6809	OS-9 Level I or II	D. P. Johnson 7655 Cedarcrest Street Portland, OR 97223	Forth-83

Table 2: The minimum list of words a Forth beginner should learn.

Arithmetic:	+	-	*
	/	MOD	/MOD
		*/MOD	
	ABS	NEGATE	
	MIN	MAX	
Comparisons:	<	=	>
Control Structure:	IF	ELSE	THEN
	DO	I	LOOP
		+LOOP	
	BEGIN	UNTIL	
Data Stack:	DROP		
	DUP	OVER	
	SWAP	ROT	
Defining Words:	:	;	
	CONSTANT	VARIABLE	
Input:	KEY		
Logic:	AND	OR	
Memory Access:	!	@	
	C!	C@	
Misc. & System:	FORGET	WORDS (VLIST)	
Output:	.	."	
	CR		
	EMIT		

OS-9 And The Macintosh

Recently I had the opportunity to test the Professional version of OS-9 for the Macintosh computer from Ultrascience. As a magazine deeply involved (more so than any other) with OS-9 and I personally being one of the oldest OS-9 users (In the early days of OS-9 each user was assigned a serial number, mine is #5) I appreciate OS-9 as being a leading-edge product and have used and supported it as long or longer than probably any other user. OS-9 for the Mac, by Ultrascience is a welcome port.

Real-Time
Multi-User
Multi-Tasking
OS-9
Fits Right Into
Mac

This is the first Mac OS that is capable of real time multi-processing & multi-user operation. I can only report to you that it works, and give you some insight of what it is like on the Mac. It is available for the Mac Plus, SE, Mac II series and SE/30. That accounts for the three currently most popular 68XXX CPUs. The 68000, 68020 & 68030. The version we tested was for the Mac II with the 68020 CPU.

Installation was the most simple OS-9 installation I have ever done. Basic requirements are a Mac with 1 meg of RAM and a hard disk with at least 5 meg of free space. You just create a folder on the Mac and drag the supplied programs from the Mac 3 1/5" drive to the Mac folder. The simplicity is a result of this version of OS-9 running as an application. The Mac OS and Finder are still in control of the hardware. While this slows the overall operation of OS-9 down somewhat, on our Mac II it is still fast. Compared to our 68020 Mustang™ from Data-Comp running Professional OS-9, the difference was not objectional but was apparent. However, it is a very well done project, mainly because OS-9 was developed and tuned to the 68XXX series of Motorola CPU devices. Meaning, this port of OS-9 is well suited for the Macintosh series of computers.

After you have copied the programs to the Mac you then go through a short series of steps to preparing OS-9 to run. You will be initializing the OS-9 hard disk (up to seven hard disk devices are supported.) 'os9setup' is then edited changing the boot file to the newly created /h0, and away you go. One thing to be aware of. If you anticipate a change or upgrade to your Mac you will have a problem as OS-9 for the Mac checks for your serial number and CPU type. If they do not match the CPU type and serial number you furnished Ultrascience it will not run. This is one of the few Macintosh applications that uses a copy protection scheme.

The floppy disk is supported only for the Mac format in the version tested. Ultrascience promises floppy support for the OS-9 format in the near future. However, Kermit is supplied so that files may be ported between the Mac and other OS-9 systems. When the floppy drivers are available we will let you know.

The Mac printer port is designated /t1 and the modem is /t2. Additional serial ports are supported. The Macintosh monitor is designated /qd1.

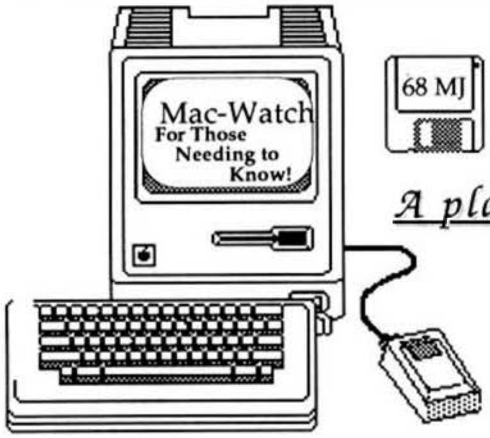
OS-9 has matured over the years to one of the most popular development and applications engines available today. It is very compatible with UNIX, especially at the 'C' source level (the full feature Microware C.) I have seen thousands of lines of UNIX C directly compiled to OS-9 without any changes. That means that most all UNIX applications can be ported to the Mac and run under MAC/OS-9 from Ultrascience. Not to mention the hundreds of applications listed in the latest Microware software catalog. Other languages supported are BASIC, Fortran, Modula-2, Forth, MUMPS, assembler and Informix software products. To just mention a few. It also supports the Mac toolbox, QuickDraw and AppleTalk. It's well suited to robotics and process control applications. Because of it's ease of porting, this should enhance the Mac as a serious multi-user, multi-tasking system. Now, what are you waiting for?

Additional information can be secured from:

Ultrascience
Gibbs Laboratories, Inc.
1824 Wilmette Ave.
Wilmette, IL 60091
Tel:312 256-0080
Fax:312 256-0097

DMW

EOF



The Macintosh™ Section

Reserved as

A place for your thoughts

And ours.......

Mac-Watch

LETRASTUDIO

Contact: Letraset USA
40 Eisenhower Drive
Paramus, NJ 07653

It isn't often that a new software product comes along that has the potential to change the way traditional desktop publishers handle display typesetting. However, a rather unique product from Letraset® has changed all that - LetraStudio.

There are several other programs that can change the way a set of characters can be modified to suit the typesetters needs, but this application has them beat by a country mile! I could spend pages attempting to tell you what all this program does, however, as some Chinese fellow once said, "A picture is worth a thousand words, or so."

The centered logo was originally a tutorial exercise in LetraStudio and was different in many respects. But, with just a couple of hours of learning LetraSudio I was able to transform it into a usable logo for our use. On the next page are a few samples that should give you an idea of the power and utility of LetraStudio. Most of the examples on the next page were done in just a few (1-5) minutes or less each.

One of the few negative, but important, things about LetraStudio is that you are constrained to using their fonts. Of which only two are included with the program. Two more fonts are sent to the buyer upon receiving the registration card, and if you happen to have bought some other of their products you might receive two or three more. However, I am not sure that policy still holds.

The price of fonts to add to your font portfolio for LetraStudio is rather steep, on the order of about $75.00 per font. So, expanding your font selection could raise the price rather sharply. However, if you have ever used some of the other font "fiddling" applications available to the Mac user, you will find that ease of use and time saved (which can be a major part of end-product cost) is well within reasonable limits. All in all, I feel that even with the additional cost, as needs arise, this is an outstanding addition to any collection of desktop publishing tools! And that my friends is from the "Granddaddy of desktop publishing."

Yes, I (and the staff here at 68 Micro Journal) are finally getting recognition as the folks who were the first to use desktop publishing commercially or otherwise. For over eleven years this magazine has been completely typeset and laid-out using traditional desktop publishing hardware and software. Years before Apple ever thought of the Macintosh or desktop publishing. It wasn't called desktop publishing then, that word was coined later. I was just the first to start using it as a complete micro-computer system, printer and all. What I started with was crude as compared to what I am using today, but it was all on the desktop and I made it commercially available (systems sold here in the USA, Canada, Mexico, South America and Europe.) Using a SWTPC kit (6800) a self-modified ball typewriter and later a daisy wheel printer and print drives and editing software I wrote (later other commercial software, computers and printers) I was able to do the first issue of 68 Micro Journal completely on the desktop, all the way from input (editing, layout, etc.) to the process of pasteup. And we still do it today! Not any other international publication can make that statement! - Now, back to LetraStudio.

The nice thing about this program is that you don't need to be an expert to use it. It is simple to learn and above all, fun. The results are truly amazing. The documentation is easy reading and very thorough.

Text can be altered in "line, envelope or straight" mode. Also text can be typed around a circle.

"Line mode" allows one or more lines of text to be distorted along the base line. "envelope mode" allows text and shapes to be grouped into an envelope and distorted together as a unit. The envelope may also be moved as a unit. Text within an envelope may not be altered. However, the envelope can be removed, text changes made and the envelope reinstalled. The circle choices are inside or outside the circle. With a combination of these functions an unlimited number of objects can be created. The examples (next page) are but a smattering of what can be accomplished in just a short time.

Another shortcoming of LetraStudio is the way that objects you create can be exported to other applications. If you use only applications that can accept an EPSF you will have no problems as LetraStudio provides an export function for that typeof file. However, it cannot open an EPSF file, so you are required to save in LetraStudio format also if you intend to ever open the file again. Text objects in straight or line mode may be saved to the clipboard and used as a normal Mac operation. However, objects that are developed in an envelope are only exported as EPSF files. This places a burden on the user who uses programs that only accept PICT or bitmapped objects. For the professional this is less a burden as he probably has and uses the more expensive applications that import EPSF files. It is hoped that this as well as the font cost problem will be addressed in an upcoming revision that is slated to be released this fall. Text files from other applications cannot be imported, leaving the user to use only LetraStudio fns

LetraStudio files may be exported to other applications and there additional text or other objects added.

LetraStudio has most of the normal text and graphic tools and functions found in other similar applications (pointer, text, zoom and graphics.) The zoom tool allows a wide range of magnification plus or minus. Graphic tools include lines, arcs, circles and rectangles. Page templates are supported. Objects (bit mapped or PICT) also may be imported and used as templates to set precisely objects being developed in LetraStudio. Grid lines may be displayed and sized, snap to grids can be turned off or on, grid/guide lines may be set to any angle for precise text alignment. The ruler can be set off or on and may also be scaled and set to inches, millimeters or points (the typesetters system of type measurement.) The ruler cannot be zeroed, which I hold a minus. Also only three sets of guidelines may be on the screen. The work area is 28 inches square, I find that more than adequate. Crop marks are supported and may be placed by the user as he desires. For those having a high resolution monitor, tonal information may be displayed, making things a lot nicer. A pref file is supported. Text kerning is supported and overlapping one side or the other is another nice feature. Objects or lines of text may be rotated by selected degrees or by the free-hand method. Many of the functions and operations are supported by keyboard input using the command and option keys in conjunction with other keys.

The distortion functions are menu derived and allow for almost endless alterations to text or envelope objects (linear scaling & distortion, horizontal and vertical, curvilinear distortions also either horizontal or vertical.) Beziers controls are supported on lines and objects. See examples.

Fills may be by percent of white, black, none or Pantone designation. Stroke lines and arcs have a choice of round, butt or projected caps. Rectangles may have round, bevel or mitre corners.

Duplications are supported to a high degree. You can produce complex effects with ease and even select number of duplications and offset amounts of each duplication, as well as order of duplication (behind or in front) in one operation. Duplication works on objects, graphic objects, lines of text and the contents of envelopes.

Printing is normally assumed to be directed to a Postscript device, however, QuickDraw printers are supported, but mostly at 72 dots per inch.

I found it to be everything advertised. And to sum it all up - "I wonder how we ever got along without LetraStudio?" For the desktop publisher, professional or otherwise, I give it my unqualified approval.

DMW

Message Passing Protocol

Compcontrol BV
Stratumsedijk 31, PO Box 193
5600 AD Eindhoven
The Netherlands

By: Ing. J.H. Brand and Ing. L. de Graaf

Standardizes Interface Between Intelligent Peripheral Controllers, Cuts VMEbus Loading

Although the most obvious and popular way of increasing the power of VMEbus systems is to add processor boards, this can cause problems unless the system architecture ;is well thought out. Just adding processor boards can crowd out the bus, and degrade system response times. For example, a VMEbus system with twenty processor boards can by definition only allow each processor an average bandwidth of 1.5 Mbytes/s. Designers cannot afford to have this banwidth taken up by routine, low-level communications.

Incorporating intelligent peripheral controllers into the system can alleviate bus loading, since data can be processed on the boards before passing it across the bus. Disk handlers can for example return sorted lists of disk contents rather than transporting large data blocks across the bus for the host processor to sort. Intelligent interface boards, industrial control, and image processing boards can all similarly increase overall system performance.

Such intelligent peripheral controllers can, however, bring their own problems. Each board requires its own driver, and designers have to write the software routines that allow processors to communicate with the controllers efficiently and reliably. If there are several peripheral controllers, this is no simple task, and strikes at the major advantage of the VMEbus itself: standardization. And without a well defined and orderly message handling procedure, designers will find it very difficult to trace the cause of any system lockups that may occur during run-time.

To solve these problems, Compcontrol BV (Eindhoven, The Netherlands see advertisment on the inside back-cover) has written a Message Passing Protocol which standardizes communications to its own range of intelligent peripheral controllers. It is now making this protocol freely available to VMEbus manufacturers and users, who can take advantage of the protocol to reduce the load on transmitting processors. It does this by concentrating arbitration and communications overheads associated with data transfers within the receiving peripheral controllers themselves, thus freeing the system processors to carry on with their own tasks.

The simple method of communication makes the protocol virtually hardware independent; the transmitting and receiving boards communicate with each other using either interrupts, or a mailbox. A transmitting processor needs only a few bytes of RAM (for storing address pointers) that can be accessed by both the processor and the VMEbus. Similarly, the receiving processor needs enough dual-access RAM to store a single message; a DMA controller can increase the efficiency of VMEbus traffic in large multiprocessing systems.

Updating The Queue

At system initialization (Fig. 1), each receiving board has a queue consisting of only two entries (the QHP and QTP). Both these entries point to the same NULL message, Which tells the receiving board that no messages are pending.

After initialization, a message is added to the queue as shown in Figure 2. A transmitting processor (MPU1) copies the existing NULL message into its own RAM (at, for example, address 11000). Then, it updates the receiving board'sCP topoint to its own message (10000), and updates the NMP to point to the new NULL message (now 11000); the pointer to the new NULL message (11000) is written at the QTP. Finally, it generates a local interrupt to inform the receiving board that the queue has been updated (the receiving board then does not need to poll the message queue). Figure 3 shows the effect of queuing another message (adding yet one more message would bring the system into the state shown in Fig. 4).

A transmitting processor does not need to wait for a message to be handled before sending a second. The only proviso is that the message queue must be updated as an individisible cycle; either the sending processor keeps control of the VMEbus for four cycles when updating the message queue, or it sets a semaphore which prevents access to the Queue Tail Pointer.

Removing a Message From The List

To remove a message from the list, a receiving board reads the message address from its Contents Pointer (address 10000 in Figure 3). It then copies the message into its own RAM, and updates the NMP anCP registers (respectively replacing 11000 with 25000, and 10000 with 20000). The receiving board then acts on the message. Finally, the receiving board sets the NMP and CP to zero to allow the memory used to store the original message to be re-used, and informs the transmitting board that it has done so.

A key feature of any message queue is what takes place when a processor tries to access the last entry in the queue (there can be problems if the processor accesses the last entry in a different manner from other entries). As shown in Figure 1, Compcontrol's Message Passing Protocol informs a receiving board that no messages are pending by the Queue Head Pointer's NMP holding a zero.

Compcontrol's MPP also has important side benefits, including simplifying system expansion. Each new transmitting processor simply needs a copy of the protocol, and the address of the Queue Tail Pointer of each receiving board it will need to address. Each new receiving processor needs a copy of the protocol, and exclusive use of enough registers to hold the pointers (the transmitting boards must also be informed of the new board's existence, and the address of these registers).

Overheads for the MPP protocol are low. The initial queue can be established using only 4 long words, with each message adding an overhead of 15 long words. The transmitting processor always accesses a receiving board's message queue in the same manner, which makes for simple and standardized software. The user can allocate his own priorities to the messages, and receiving boards can generate time-outs if a message is not handled within a predefined period.

A peripheral controller can act as either a receiving or (if it has access to enough memory space for storing messages) a transmitting board. Messages can be stored either in RAM on the transmitting board itself, or on disk somewhere in the system until the receiving board is ready to act on them. The messages may be on any length; any number of processors can access a receiving board, and there is no limit to the number of messages which can be queued; limits are set only by the amount of available memory to store the messages.

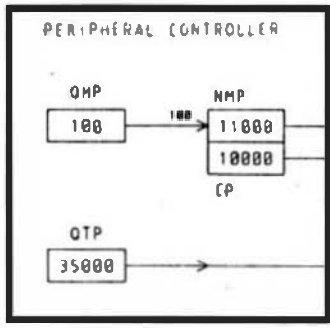
Each receiving board has its own message queue, consisting of a header, command block, data block, and status block. The fixed-size header gives the receiving board information such as the mailbox address (or alternatively which interrupt level and interrupt vector) it should respond to when the message has been handled. The variable-size command and data blocks are followed by the status block, which tells the board which action it must take if it cannot act on the message.

A system's message queue is distributed around the transmitting boards. Each queue entry has two 32-bit words: one holds the address of the current message (the Contents Pointer, CP), and the other holds the address of the next queue entry (the Next Message Pointer, NMP). These queue entries therefore form a linked list; the final queue entry (called the NULL message) has an NMP of 0 and a CP of -1 (an inaccessible address which indicates that there is no associated message).

The only other reference addresses which the system needs to hold are the 32-bit addresses of the queue head (the Queue Head Pointer, QHP) and queue tail (the Queue Tail Pointer, QTP). These are both stored on the receiving boards.

The protocol can even help here, since it can be used to teach peripheral controllers new commands. A transmitting processor could, for example, load application-specific software onto a newly-installed recieving board using only a single instruction: "learn new command." MPP also makes is very simple to trace the cause of the system lockups which are inherent to multiprocessing systems. Such lockups occur when one program is waiting for the results of another program, which in turn is waiting for the results of the first program. With MPP, the relevant registers show all the system's pending messages.

More information on the protocol (including a protocol description) is available from Compcontrol BV, Stratumsedijk 31, PO Box 193, 5600 AD Eindhoven, The Netherlands.



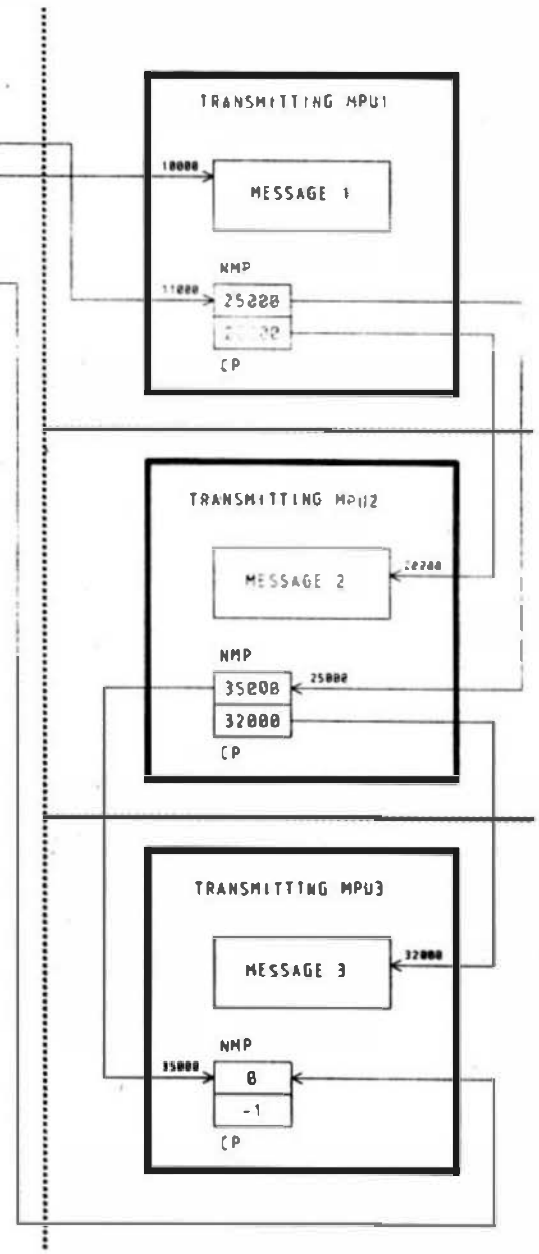


Figure 1: When a system is initialized, the receiving board's Queue Head Pointer (QHP) point to the same location (a NULL entry), which shows that no messages are pending.

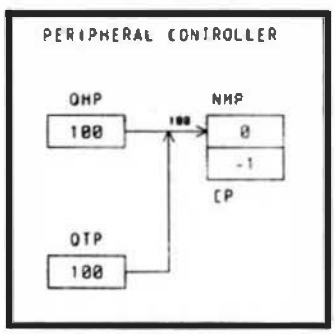


Figure 2: To add a messages to the queue, a transmitting board copies the null message to a new location and updates the Next Message Pointer (NMP), Contents Pointer (CP) with their new values. Then, it updates the receiving board's Queue Tail Pointer.

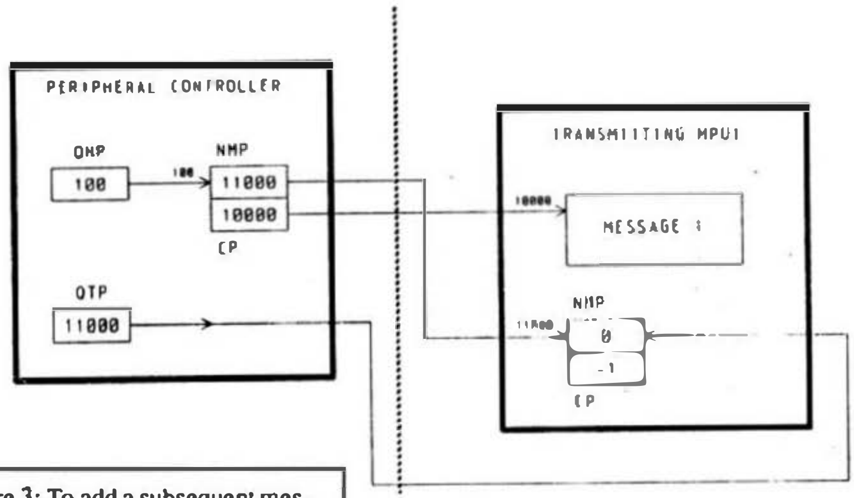


Figure 3: To add a subsequent message to the queue, the procedure is the same as that shown in Figure 2.

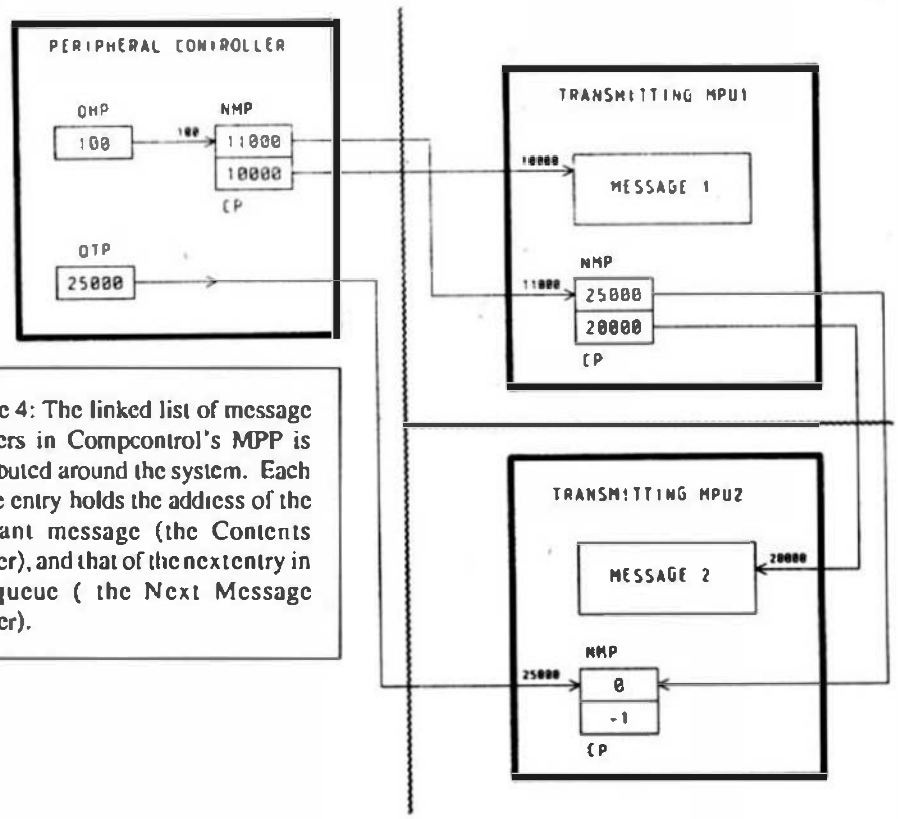


Figure 4: The linked list of message pointers in Compcontrol's MPP is distributed around the system. Each queue entry holds the address of the relevant message (the Contents Pointer), and that of the next entry in the queue (the Next Message Pointer).

CAREEN 68K brings the power of OS-9/68000 in your IBM-PC

The high performance Add-In-Board **CAREEN 68K** changes the character of your PC from a simple documentation tool to a powerful real-time software development system.

If you want more informations, here is your contact address:

LP Elektronik GmbH
Ettishofer StraBe 10c
D-7987 Weingarten
West Germany

Telefax: (0751) 53199
Telephone: (0751) 52327

PROFILER

Discover which functions within your C or Assembly programs take the most time to run. An invaluable development tool for anyone worried about program speed.

For information on this product and our other OS-9 software development packages: IMP (the Intelligent Make Program), STIMULUS (Artificial Intelligence Language), PAN UTILITIES and WINDOWS (source code library) contact:-

PAN CONTROLS LIMITED,
Drummore, Doune,
Perthsbire,
FK16 6AX, SCOTLAND.
Phone (+44) 78685-261.

ADICON is a consulting and engineering firm providing a range of services, including product specifications, design and development, fabrication of prototypes, software, and complete documentation for OEM's. We specialize in the following MOTOROLA products:

HARDWARE DEVELOPMENT
- 6805, 68HC11, 68000/20
- Single Board Computers
- VMEbus systems

SOFTWARE DEVELOPMENT
- C
- Assembly Language
- UNIX

ADICON CONSULTING AND DESIGN
1123 N. Water St., Milwaukee, WI 53202
414/276-6800

- HARDWARE DESIGN, BREADBOARDING, AND PROTOTYPING
- SOFTWARE DESIGN, PROGRAM ASSEMBLY, AND FIRMWARE IMPLEMENTATION
- INTERFACE DESIGN AND DEVELOPMENT
- MICROCOMPUTER BASED MEASUREMENT AND CONTROL
- DESIGN OF SINGLE CHIP MICROCOMPUTER BASED PRODUCTS A SPECIALTY

MICRODYNAMICS
P.O. BOX 2716
WARMINSTER PA 18974 (215)-357 6805

Fulfilling the needs of the nation's high tech community for over a decade.

Leaders in Hardware and Software consulting for real-time military and industrial applications.

AI MICROSYSTEMS
AI Microsystems, Inc.
One Naperville Plaza
Naperville, IL 60540
(312) 416-2177

Tomorrow's Technology Today!

RADON

Use Your Computer To Measure Radiation. Write Or Call For Our Sensor Catalog.

For more information please call or write:
LaGrange Instruments, Inc.
Kuchler Drive
LaGrangeville, NY 12540
(914) 223-3336

ATARI ST

OS9 Professional™ incl:
C, Basic, Emacs, Kermit,
S record download, OS9 to TOS transfer,
Sculptor™, Dynacalc™, Stylograph™

8 Serial Port board for the MEGA ST.
12 bit Analogue I/O interface.
16 bit Parallel Interface.
EPROM Programmer.

UNISON,
T.J.P. Electronics Ltd.
3 West Street,
Scarborough,
North Yorkshire,
ENGLAND. U.K.
YO11 2QL

Tel U.K. 723-378837
FAX U.K. 723-500435

COMPUTER PEREPHERIALS
FOR THOSE WHO
LIKE TO BUILD
THEIR OWN.
FOR MORE
INFORMATION
SEND A S.A.S.E.
TO

HARLEEN FRANCISCO
8332 PEGGY ST.
TAMPA FL.
33615

XYZ
ELECTRONICS INC.
A NAME YOU'LL REMEMBER
ROUTE 12 BOX 322
INDIANAPOLIS, IN 46236
(317) 335-2128

First to offer OS-9 68000 on the STD Bus. Full line of memory, serial, parallel, analog, digital, and video I/O. Full systems too!

Classifieds

As Submitted - No Guarantees

SWTP S/09 - 384 K, 10Meg Hard Disk, 8" Floppy, 2-X-12 Word Processing Terminals, Furniture, Software. Paid $9000 in 1983, Stored since '84.
Best Offer (618) 667-2544

***

PT68008 Single Board Computer, 10 MHz, 2 RS-232 Serial Ports, 768 K RAM/64K Eprom, 1-5" Drive, 20 Meg Hard Disk, 2-8 Bit Parallel Ports, OS9, Stylo Pak, Sculptor. $1495

CDS--1, 20 Meg Hard Disk w/controller $100
S+ Memory Cards, CPU Cards, Hard Disks w/ Controller Cards, I/O Cards, Cabinets, Power Supplies.
S/09 CPU Cards, Memory, I/O Cards, Controller Cards
5-Siemens 8" Disk Drive, $100 ea
Tom (615) 842-4600 M-F 9AM to 5PM EST

***

THE 6800-6809 BOOKS

OS - 9 User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles

OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9

HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building anew System Disk, OS9 Users Group, etc.

Programming Languages

Assembly Language Programs and Interlacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include

No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset — w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder $5.50

All Source Listings on Disk

1-8" SS, SD Disk $14.95
2-5" SS, SD Disks $24.95

FLEX USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XXusers, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the TEXT files included in the book and on diskette. All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory - ASM PIC
MEMOVESC1	Memory move program - ASM PIC
DUMP.C1	Printer dump program - uses LOGO - ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences - ASM
TERMEM.C2	Modem input to disk (or other port input to disk) - ASM
M.C2	Output a file to modem (or another port) - ASM
PRINT.C3	Parallel (enhanced) printer driver - ASM
MODEM.C2	TTL output to CRT and modem (or other port) - ASM
SCIPKG.C1	Scientific math routines - PASCAL
U.C4	Mini-monitor, disk resident, many useful functions - ASM
PRINT.C4	Parallel printer driver, without PFLAG - ASM
SET.C5	Set printer modes - ASM
SETBAS1.C5	Set printer modes - A-BASIC

Note: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

** Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: $7.95 + $2.50 S/H

With disk: 5" $20.90 + $2.50 S/H
With disk: 8" $22.90 + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

MASTER CARD

VISA

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.

Hixson, TN 37343

Telephone (615) 842-4601
FAX (615) 842-4607

68' Micro Journal
Reader Service Disks

Disk-1	Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk-2	Diskedit w/ inst.& fixes, Prime, *Prnod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk-3	Cbug09. Sec1, Sec2. Find. Table2. Intext, Disk-exp. *Disksave.
Disk-4	Mailing Program. *Finddat, *Change, *Testdisk.
Disk-5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN. **BLACKJAK, **BOWLING.
Disk-6	**Purchase Order, Index (Disk file indx).
Disk-7	Linking Loader, Rload, Harkness.
Disk-8	Crtest, Lanpher (May 82).
Disk-9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc,
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong),CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat. PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov.85. Extensible Table Driven. Language Recognition Utility, Anderson Mar86.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - Feb .87
Disk-33	Build the GT-4 Terminal - Nov. 87 Joseph Condon.
Disk-34	FLEX 6809 Diagnostics, Disk Drive Test, ROM Test, RAM Test - Apr. 89Korpi.
Disk-35	DO A FLEX-09 Batch File Processor - Oct. 88 - Dave Howland
Disk-36	Add Graphics To Your SBC - Nov. 88 - Joseph Condon

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC

*** Denotes 68000 - 6809 no indicator.

8" disk $19.50

5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50

Overseas add: $4.50 Surface - $7.00 Airmail

68' MICRO JOURNAL

5900 Cassandra Smith Rd.

Hixson, TN 37343

(615) 842-4600

FAX (615) 842-4607

VISA

MasterCard

SOFTWARE

68000 C CROSS-COMPILER
$100 - SKDOS,MSDOS,UNIX,XENIX (OBJECT ONLY)

Accepts K&R C language, generates 68000 assembler code
includes 68010 cross-assembler, libraries provided for SKDOS, but may be modified

CROSS-ASSEMBLERS WITH MACRO CAPABILITIES
EACH $50-FLEX,OS9,UNIFLEX,MSDOS,UNIX,SKDOS,XENIX 3/$100 ALL/$200

Specify: 180x, 6502, 6801/11, 6804, 6805, 6809, Z8, Z80, 8048, 8051, 8085, 68010, 32000
Modular cross-assemblers in C, with load/unload utilities. Sources for additional $50 each, $100 for 3, $300 for all

CMODEM TELECOMMUNICATIONS PROGRAM
$100-MSDOS,SKDOS,UNIX,FLEX,OS9,XENIX,UNIFLEX OBJECT-ONLY: EACH $50

Menu-driven with terminal mode, file transfer, MODEM7, XON-XOFF, etc.

SUPER SLEUTH DISASSEMBLERS
EACH $99-FLEX $101-OS9 $100-UNIFLEX OBJECT-ONLY: EACH $50 FLEX,OS9,COCO

Interactively generate source on disk with labels, includes xref, binary editing
Specify 6800,1,2,3,5,8,9/6502 version or Z80/8080,5 version
COCO DOS available in 6800,1,2,3,5,8,9/6502 version (not Z80/8080,5) only
68010 version $100-FLEX,OS9,UNIFLEX,MSDOS,UNIX,SKDOS,XENIX

DEBUGGING SIMULATORS FOR POPULAR 8-BIT CPUS
EACH $75-FLEX $100-OS9 $80-UNIFLEX OBJECT-ONLY: EACH $50-COCO FLEX, COCO OS9

Interactively simulate processors, includes disassembly formatting, binary editing
Specify for 6800/1, (14)6805, 6502, 6809 OS9 only, Z80 FLEX only

ASSEMBLER CODE TRANSLATORS FOR 6502, 6800/1, 6809
6502 to 6809 $75-FLEX $85-OS9 $80-UNIFLEX
6800/1 to 6809 & 6809 to pos-ind. $50-FLEX $75-OS9 Only $60-UNIFLEX

FULL-SCREEN XBASIC PROGRAMS with cursor control AVAILABLE FOR FLEX, UNIFLEX, AND MSDOS

Display Generator / Documenter	$50 w/source,	$25 without
Mailing List System	$100 w/source,	$50 without
Inventory with MRP	$100 w/source,	$50 without
Tabula Rasa Spreadsheet	$100 w/source,	$50 without

DISK AND XBASIC UTILITY PROGRAM LIBRARY
$99-FLEX $30-UNIFLEX/MSDOS

Edit disk sectors, sort directory, maintain master catalog, do disk sorts, resequence some or all BASIC program, xref BASIC program, etc. non-FLEX versions include sort and resequencer only

PROFESSIONAL SERVICES
FOR THE COMPUTING COMMUNITY

CUSTOMIZED PROGRAMMING

We will customize any of the programs described in this advertisement or in our brochure for specialized customer use or to cover new processors; the charge for such customization depends upon the marketability of the modifications.

CONTRACT PROGRAMMING

We will create new programs or modify existing programs on a contract basis, a service we have provided for over twenty years; the computers on which we have performed contract programming include most popular models of mainframes, including IBM, Burroughs, Univac, Honeywell, most popular models of minicomputers, including DEC, IBM, DG, HP, AT&T, and most popular brands of microcomputers, including 6800/1, 6809, Z80, 6502, 680x0, using most appropriate languages and operating systems, on systems ranging in size from large telecommunications to single board controllers; the charge for contract programming is usually by the hour or by the task.

CONSULTING

We offer a wide range of business and technical consulting services, including seminars, advice, training, and design, on any topic related to computers; the charge for consulting is normally based upon time, travel, and expenses.

Computer Systems Consultants Inc.

1454 Latta Lane
Conyers, Georgia 30207
(404) 483-4570 • (404) 483-1717

Contact us about catalog, dealer, discounts, and services. Most programs in source: give computer, OS, disk size. 25% off multiple purchases of same program on one order. VISA and MASTER CARD accepted. Add GA sales tax (if in GA) and 5% shipping. (UNI)FLEX tm Technical Systems Consultants; OS/9 Microware; COCO Tandy; MSDOS Microsoft; SKDOS Stark Software

ADVANCED 32 BIT SINGLE BOARD COMPUTER

20MHZ MC68030

20MHZ 68881

4-8 MBYTES RAM

4 RS-232 SERIAL PORTS

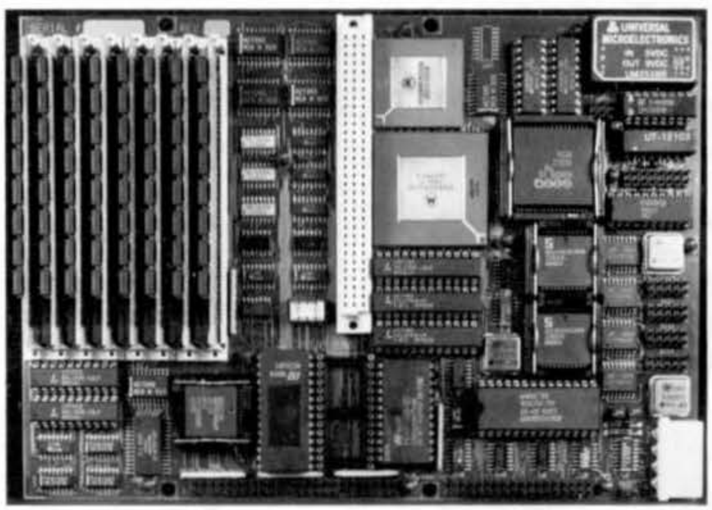
4 MBYTE/SEC SCSI

32 BIT HIGH SPEED EXPANSION

ON BOARD ETHER-NET

REAL-TIME UNIX COMPATABLE OS

$1,995 QTY 1



The RAVEN™ SBC from Cogent Engineering represents a new level of integration and performance. Contained on a single 5.75" x 8.00" board are all of the functions needed for a complete 32 bit networking system. Compare the RAVEN to other single board solutions that offer less performance for more money and you'll see why the RAVEN is the right choice for all your embedded and multi-user requirements. Call or write:
COGENT ENGINEERING 45 LAKESIDE AVE, #20
MARLBORO, MA 01752 (800) 282-7642, (508) 624-6447

RAVEN OPTIONS

8 Mbyte upgrade	$395.00
OS/9©	$CALL
UNIFLEX/RN©	$495.00
Development systems	$CALL

LOOKING FOR COMMUNICATIONS?

Then take a look at our range of quality VMEbus communications boards for the shop floor and the office. We can link systems using high-performance X.25, RS-232/422/485 Arcnet and IBM Token Ring connections using wire or optical fiber.

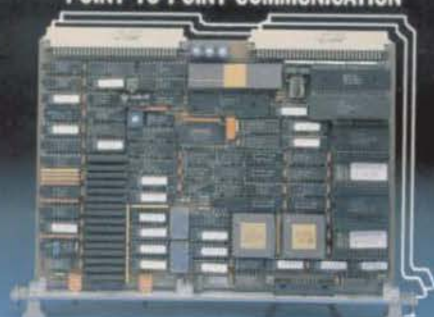
And we won't just sell a board and leave you to get on with it.
Our support department will help you to handle any special-to-application problems. The result? A working system with the minimum hassle.

Test our commitment to communications. Make a start by asking for our Microsystems file.

LOCAL AREA NETWORK

CC140 or CC-96/103 IBM TOKEN RING IEEE 802.5
4 or 16 Mbits/s

POINT TO POINT COMMUNICATION



CC-97/CC135 RS-232/422/485/fibre optic

LOCAL AREA NETWORK

CC121/CC123 Arcnet interface

WIDE AREA NETWORK

CC-125 X.25 interface

More products in your
MICROSYSTEMS FILE

ARCNET™ is a registered trademark of the Datapoint Corporation. IBM® is a registered trademark of International Business Machines.
• All modules OS-9 supported.
OS-9 is a trademark of Microware.

VITA

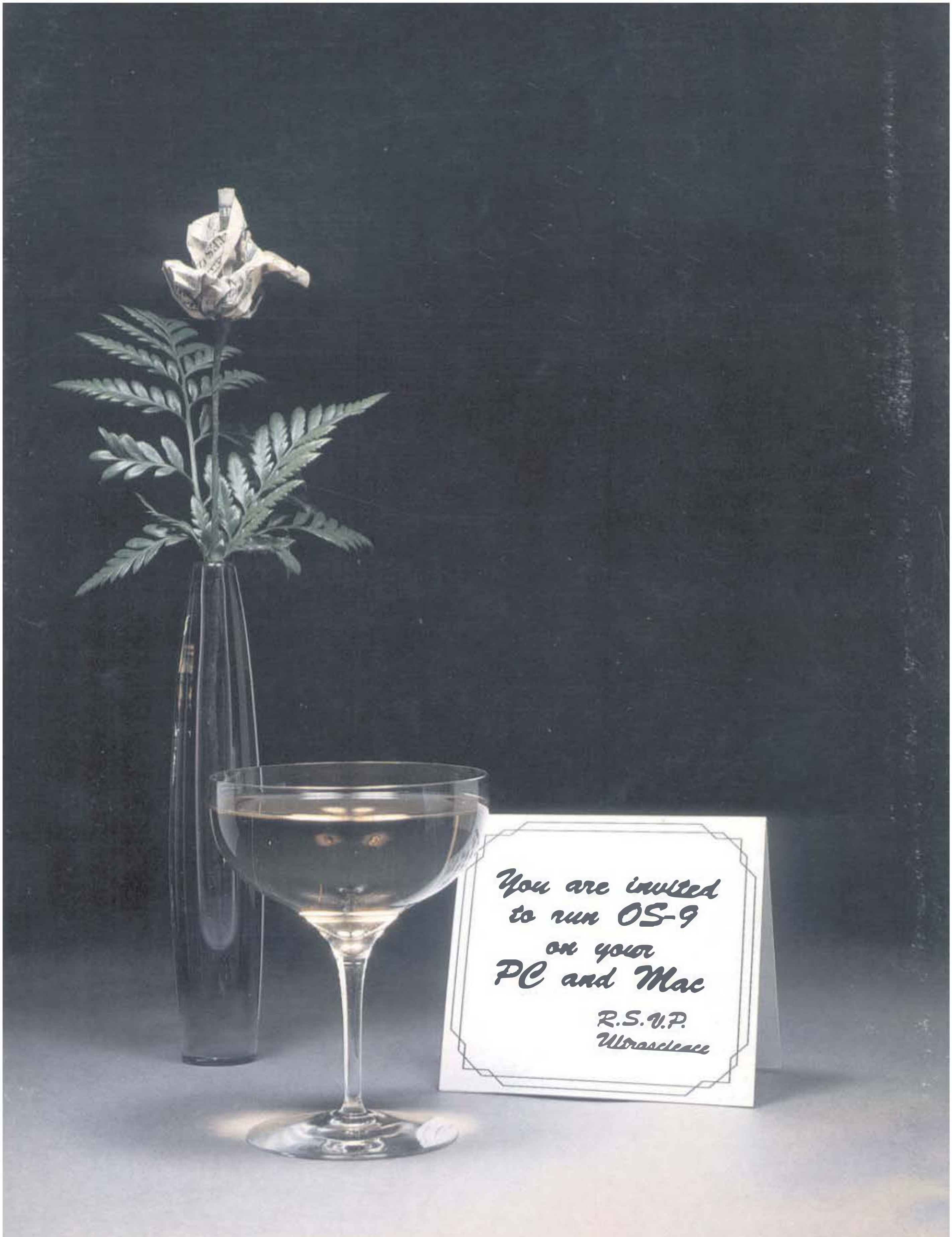
VME FUTUREBUS+
COMPCONTROL
OPEN ARCHITECTURE COMPUTERS

COMPCONTROL INC.
15466 Los Gatos Blvd.
Suite 109-365
LOS GATOS,
CALIFORNIA 95032 USA
Phone: (1)-408-356-3817
Fax : (1)-408-356-1755

COMPCONTROL SARL
148 Av. d'Italie
75013 PARIS
France
Tel : (33)-(1)45.80.84.48
Fax: (33)-(1)45.80.81.63

COMPCONTROL GmbH
Wilhelmshöher Allee 259
3500 Kassel,
Germany
Tel : (49)-(0)561-35006
Fax: (49)-(0)561-35007

COMPCONTROL B.V.
Stratumsedijk 31,
P.O. Box 193
5600 AD Eindhoven,
Holland
Tel : (31)-(0)40-124955
Fax: (31)-(0)40-120296


You are invited
to run OS-9
on your
PC and Mac
R.S.V.P.
Ultrascience